NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

UTILIZATION OF A VIRTUAL ENVIRONMENT FOR COMBAT INFORMATION CENTER TRAINING

by

John J. Kapp
and
Erkan Akyuz

March 1997

Thesis Advisor: Michael Zyda
Thesis Co-Advisor: John Falby

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1997	Master's Thesis

4. TITLE AND SUBTITLE
Utilization of a Virtual Environment for Combat Information Center Training

5. FUNDING NUMBERS

6. AUTHOR(S)
Kapp, John J.
Akyuz, Erkan

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Recent fiscal and personnel cut-backs have placed significant restrictions on surface ship training opportunities. As a result, additional methods of training must be established in order to maintain current operational readiness. This thesis research investigates the use of a workstation-based shipboard virtual environment (VE) as complementary training for naval personnel, in particular, in the combat information center (CIC).

The approach taken was to extend the Naval Postgraduate School's Shiphandling Training Simulator (SHIPSIM) and shipboard Virtual Environment Trainer to include a combat information center virtual environment system (CICVET). Using the NPSNET IV framework, the system provides two levels of training; the first reflects the dynamics of real-world warfare theaters with the capability for distant entities to interact, while the second allows for the team training of shipboard personnel, possibly in separate locations, within the same virtual CIC.

To achieve our goal we built a real-time, distributed, interactive shipboard environment for combat information center training. It consists of a three-dimensional CIC model, containing functioning consoles for information display, sensor management, and weapons control.

14. SUBJECT TERMS
NPSNET, DIS, real-time, 3D, visual simulation, network, distributed, Performer, interactive, virtual world, PVS, Potentially Visible Sets, terrain database, Voice Recognition, mounting humans entities

15. NUMBER OF PAGES
92

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Approved for public release; distribution is unlimited.

UTILIZATION OF A VIRTUAL ENVIRONMENT FOR COMBAT INFORMATION CENTER TRAINING

John J. Kapp
Lieutenant, United States Navy
B.S.C.S., United States Naval Academy, 1991

Erkan Akyuz
Lieutenant (jg), Turkish Navy
B.S.E.E., Turkish Naval Academy, 1991

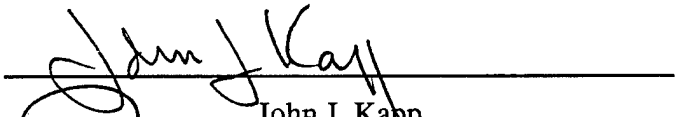
Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

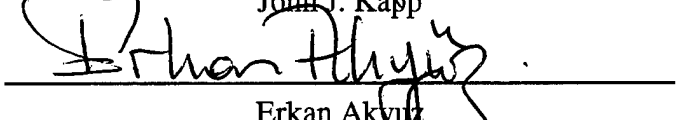
from the

NAVAL POSTGRADUATE SCHOOL
March 1997

Authors:



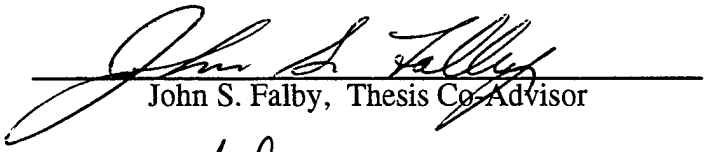
John J. Kapp



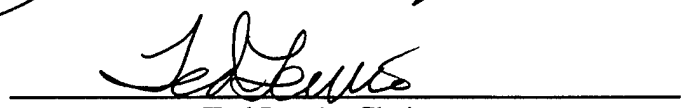
Erkan Akyuz

Approved by:

Michael J. Zyda, Thesis Advisor



John S. Falby, Thesis Co-Advisor



Ted Lewis, Chairman,
Department of Computer Science

ABSTRACT

Recent fiscal and personnel cut-backs have placed significant restrictions on surface ship training opportunities. As a result, additional methods of training must be established in order to maintain current operational readiness. This thesis research investigates the use of a workstation-based shipboard virtual environment (VE) as complementary training for naval personnel, in particular, in the combat information center (CIC).

The approach taken was to extend the Naval Postgraduate School's Shiphandling Training Simulator (SHIPSIM) and shipboard Virtual Environment Trainer to include a combat information center virtual environment system (CICVET). Using the NPSNET IV framework, the system provides two levels of training; the first reflects the dynamics of real-world warfare theaters with the capability for distant entities to interact, while the second allows for the team training of shipboard personnel, possibly in separate locations, within the same virtual CIC.

To achieve our goal we built a real-time, distributed, interactive shipboard environment for combat information center training. It consists of a three-dimensional CIC model, containing functioning consoles for information display, sensor management, and weapons control.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

I. INTRODUCTION

A. MOTIVATION

1. The Plight of Naval Training: Resources vs. Responsibilities

Shipboard training has long been acknowledged as the key to the operational success of naval units-- an undebatable point. However, the question of which training methods produce the best results has at one time or another been a hot topic in every wardroom, chief's mess, and berthing. While history may show that the old ways produced satisfactory results, Navy-wide training is still undergoing numerous transformations in an effort to perfect the process. This is not an easy task. The shipboard environment could not be described as the most conducive setting to learning. For executive officers, ensuring adequate training time for the numerous shipboard teams (Damage Control Training Team, Engineering Casualty Control Training Team, Combat Systems Training Team, Navigation/Seamanship Training Team, etc.) can be a nightmare and makes for some interesting PB4T (Planning Board For Training) meetings, as team leaders jockey for their share of the calendar. Compound this with scheduling equipment down-times for maintenance, arranging ship visits, charting shipboard inspections, while simultaneously considering the ship's underway schedule and operational requirements. Cost of current training is also a major factor. Equipment must be up and fully functional for training to be substantial, while additional underway time may be necessary. Accounting for all these factors may result in low quality and inadequate training.

Take for example, the training of the NGFS (Naval GunFire Support) team. While inport, current training involves canned scenarios where sheets of bearing and ranges (painstakingly created and checked ahead of time) are used for the team to act on. Not the most effective way to conduct training. When the ship does finally get its turn on an actual firing range, training is, of course, at peak levels. But how many times has the fog on San Clemente Island prevented the day's shoot, or how many guns and radar units have

malfunctioned when call for fire is given? This real-time environment training is at a premium and lost opportunities are not easily replaced.

The recent drawdown of personnel and the ancillary dwindling of military budgets, without a complementary and proportionate reduction in operational requirements, has placed great burden on today's leaders. In fact, the increasing concern over budgetary matters means that Commanding Officers must seek out enterprising new ways to train their crews if the ship is to meet its challenges successfully. The monetary costs associated with underways for training or the time costs connected with scheduling simulator time prohibit flexibility in engineering professional training evolutions and may even prevent traditional means of conducting training. Additionally, the dwindling of realistic training periods and the diminishing underway time result in "book smart" sailors, whose training records will lack the necessary "hands-on" experience required to fulfill their duties professionally, competently, and safely. What is needed is an innovative change to training that can encompass the best qualities of at-sea and simulator training while defraying the associated costs and alleviating the problems inherent to current inport training methods.

2. Virtual Environments

Training at sea and training in large scale simulators share the goal of immersing trainees into an environment that closely approximates the actual environment the personnel will be expected to perform in. This is so, not only because learning is at a maximum when the student is surrounded by the sights, sounds, and contacts of realistic environments, but also because intangible experiences such as those brought about by working in stressful situations, cannot be learned sitting at a desk. Simulators accomplish their goal through the use of virtual environment technology.

"VE technology developments over the past decade have produced the ability to synthesize large scale, three-dimensional models, such as a Navy ship, and produce a real time, interactive simulation. Advances in network systems allow these graphical simulations to communicate, enabling a large number of people to interact in the same

virtual environment. Although these virtual environments can be created, their capacity to solve the Navy's training problem must be demonstrated before building them makes sense" [MCDO95]. Virtual environments provide users with a sense of "presence," or the feeling that the environment is occupied and not simply looked at. Figure 1 is an illustration of this notion.

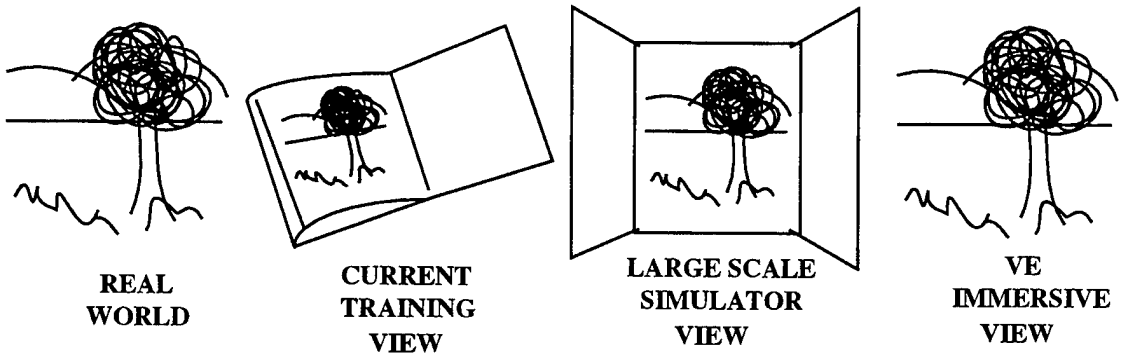


Figure 1: Virtual Environment Immersive View vs. Current Training View

For users surrounded by the environment, there is no need for altering their natural perceptions in order to absorb what is happening around them. For instance, no need exists for imagining environmental factors that cannot be represented in two-dimensional form. Thus, personnel can interact within this virtual environment using the same natural semantics that they use when interacting with the physical world, which means that they are more likely to take lessons they learn there into the real world [HITL94]. "It is the sense of immersion and inclusion in the virtual educational environment that may allow the student an opportunity to interpret and encode his or her perceptions and paradigms from a broader, deeper set of experiences than those that can be had in the "standard" educational environment."[OSBE92]

VE technology is ideally suited to safely prepare Navy personnel for the complex environments they encounter on Navy ships. In recent years the Navy has demonstrated the viability of using virtual environments for training. Incorporating as much of the actual hardware to be trained on as possible, large simulators, such as for ship-handling or flight training, are currently proving their worth in the fleet. Their major drawback is the fact that

there is a limited number available to serve the entire fleet and their fixed location make their use by any but the co-located ships impractical. A solution is to model all aspects of the real-world into a virtual environment that is cost effective, portable, adaptable, and expandable. Virtual environments can produce a training environment far more realistic than methods currently in use aboard ships, at a cost far below that of shore based trainers, and they are able to be used by the ship wherever it goes.

To continue the previous example, suppose the NGFS team had the capability to practice their skills in a virtual environment. The NGFS team wouldn't have to worry about any malfunction or weather problems. Meanwhile, their immersion within the environment would give the same training as would an actual shoot. Additionally, other networked players in the scenario, such as tank drivers or infantry, could be the actual units requiring the support. Meanwhile the ship's helo, another environment entity and piloted by an actual pilot, could provide third party targeting and bridge watchstanders could practice coastal navigation. The virtual environment gives the team the capability to perform operations that were once limited to just evolutions conducted at sea. Now qualifications and maintenance of qualifications can be performed without the ship ever slipping its mooring. Total integration of the team into the operational capability of the ship is the goal.

B. BACKGROUND

Investigating new opportunities in virtual environments and applying tomorrow's technology is the goal of NPSNET. NPSNET is a collaboration of faculty, staff and students to demonstrate the feasibility of a low-cost, large-scale, distributed virtual environment. Using commercial off-the-shelf components, it utilizes the DIS (Distributed Interactive Simulation) communications protocol for these multi-platform simulations. By being distributed, units in separate locations can train together, interacting with each other in the virtual environment.

NPSNET IV is the most current version of the evolving NPSNET simulation system. "The architecture uses BSD 4.3 socket-based interprocess communication (IPC) to

provide a clear, easily used, and well-documented network interface. NPSNET IV, the simulation application, is tailored with efficient mechanisms to map DIS data to NPSNET data structures. The DIS network library that was developed through previous research at NPS resulted in a network harness for DIS applications. NPSNET further provides a three dimensional virtual world (VW) on Silicon Graphics workstations" [ZESW93].

Previous work with NPSNET has shown the viability of using a virtual environment for shipboard training. NPSNET also shows that the same training accomplished by larger simulators can be done in a low-cost, workstation based environment. While focusing on the fundamental three-dimensional nature of team training environments and of human perception, virtual environments using NPSNET have been shown to be excellent means for both shiphandling and damage control [MCDO95] [OBYR95]. The success achieved here drives the idea of extending the research into other areas, particularly combat systems. This research is an attempt to expand the scope of NPSNET's current implementation, which is primarily geared toward ground-based forces and operations, to include naval assets as weapons platforms.

C. OBJECTIVE

The objective of this thesis is to augment the existing NPSNET naval asset with a functional Combat Information Center (CIC) and a weapons capability to produce a ship virtual environment for combat systems training. This environment encompasses the best attributes found in real-life and simulator training while offering new advantages of onboard remote operation feasibility and portability. The prototype produced is of an interactive, real-time, networked, virtual Combat Information Center which can be installed on board Navy ships. This prototype can then be evaluated for its effectiveness in training personnel. Additionally, the prototype allows for multiple users to participate within the same scenario or environment. This is key to proper team training. At the same time, multiple users enhances the concept of apprenticeship training. Recruits receiving

their introduction to spaces and equipment can enter the virtual environment as an observer and see first hand the operations and functions they will be expected to perform.

The CIC trainer is representative of a real ship's CIC. It is built to generically simulate actual function and operation of a ships radar and weapons control center. This virtual environment is meant to augment current training methods by offering environment familiarization and new skills gained through the experience of interacting and operating with the virtual environment.

D. SUMMARY OF CHAPTERS

The remainder of the thesis is broken down as follows:

- Chapter II comments on previous research in developing virtual environments, particularly NPSNET.
- Chapter III offers a system overview of the CICVET ship project.
- Chapter IV discusses the development tools used to build CICVET.
- Chapter V details the implementation of large screen displays.
- Chapter VI presents the approach taken to create a functioning radar within the virtual environment.
- Chapter VII describes the construction of a weapons interface panel.
- Chapter VIII explains the alterations made to NPSNET weapon launch and flight.
- Chapter IX provides a final discussion of the results of this thesis and describes follow-on work to be accomplished.

II. PREVIOUS RESEARCH

This chapter briefly examines the research efforts made in the development of virtual environments at the Naval Postgraduate School. Further, it focuses on the distributed computing issues involved with building NPSNET, in particular the naval ship entity.

A. NPSNET

NPSNET is the fundamental platform by which faculty, staff, and students of the Naval Postgraduate School implement various areas of research in networked virtual environments. Implemented on commercial-off-the-shelf SGI (Silicon Graphics IRIS) workstations, it is designed to show that multi-player networked battlefield simulation is feasible for both training and evaluations. Users can inhabit and interact within the simulation over local area networks or multicast over the Internet's MBONE (Multi-cast Backbone). Originally intended for use in Army simulations, recent developments include creation of naval assets and continued research is being steered toward naval training.

NPSNET communicates using the DIS (Distributed Interactive Simulation) network protocol, allowing communication with other simulators that also use the DIS standard. DIS evolved from SIMNET (Simulation Network), a distributed interactive simulations standard developed by DARPA in 1985 [LOCK94]. It utilizes many types of Protocol Data Units (PDU) to transmit information between simulators. Each PDU is used for a specific purpose, and encompasses all aspects of a simulation. The DIS PDU's allow the developing events of a simulation to be shared among different host sites.

B. MCDOWELL and KING: Virtual Shipboard

In 1995, Perry McDowell and Tony King showed with their thesis that a prototype shipboard virtual environment was feasible [MCDO95]. They constructed a ship model based on the Antares, a roll-on/roll-off ship. By storing the model in a hierarchical data structure and utilizing an algorithm for determining potentially visible sets, they

successfully demonstrated that a large model database representing a ship could be visualized at real-time, interactive frame rates.

The choice of which model to use was driven by several factors, the most important ended up being availability. In an ideal world, models of actual naval vessels would be used for rendering within the simulation. Seemingly, this process would entail converting CAD (Computer Aided Design) data of navy platforms into a polygonally-based database that could be understood and rendered by three-dimensional visualization software. McDowell and King could not obtain such data in time for their study. Thus, they opted to utilize data from the Antares, a non-warfare platform that was under design. This model served as the basis for implementing environments that would be representative of warships. Initially, the model consisted of approximately 2,000 polygons but after interior spaces, such as a bridge, an engineering space and a combat information center, were added, the database increased in size by over a factor of 10. Using the Multigen modeling tool, McDowell and King transformed the ship into a hierarchical data structure that was consistent with Silicon Graphics' IRIS Performer, the rendering software of SGI workstations[MCDO95].

In order to provide as realistic an environment as possible, various techniques were used to maximize performance of the database. Combining both Multigen's and Performer's level of detail features shrunk the rendering time by offering alternatives for visible objects in a scene. These alternatives provided different aspects of an object based on factors such as how far from the viewpoint were the objects. The further away the less detail need be included in the object. Another enhancement method utilized was instancing. This is the ability to maintain only one geometry for an object in memory even though there may be greater than one of these objects in the scene. Thus, performance increases as the additional copies are not loaded into memory. Realism was boosted by use of texturing, dynamic objects, and switch objects. Texturing entails mapping actual scanned images onto the geometry in the scene to make objects appear as they do in the real world. Both dynamic objects and switch objects are attributes of the Performer hierarchy which allow scene motion and object removal, addition, and replacement.

Once realistic interior spaces were developed, research turned to showing how the environment could be suited for training. In the engineering space, functionality for representing casualties was implemented-- one a steam leak and another a fuel oil leak/fire. Users in the space, represented by a non-articulated human entity, could manipulate fire-fighting equipment, such as halon discharge, nozzles, and ventilation controls, or other objects, such as steam/fuel oil valves. The point demonstrated was that a virtual environment could simulate casualties with more realism than current shipboard methods, such as red flags representing fire.[MCDO95]

An additional feature for training was the path walk-thru. This was meant to show how the virtual ship could be used as a familiarization for newly assigned personnel, thus eliminating much of the time these crewmembers would require simply acquainting themselves with the ship.[MCDO95]

Additionally, the VE (virtual environment) was networked allowing for multiple users to inhabit the world and interact with each other. Features such as an elaborate collision detection scheme, environmental effects (smoke from fires), HMD (head-mounted display) inter-operability, and various training scenarios were implemented in order to provide an extensive shipboard atmosphere. The research left many areas open for future work, including functionality of other spaces, with the combat information center of particular import to this thesis.

C. O'BYRNE: DC VET

James O'Byrne followed up on the work of McDowell and King. His goal was to expand the casualty functions of the engineering spaces into a full-fledged Damage Control Virtual Environment (DC VET). By improving the realism, the DC VET offers a new training method that complements or supplants current methods. Providing a bridge between the gap in school training and performance aboard ship was the objective.

The DC VET was designed to train damage control teams in as realistic an environment as possible. To this end, embellishments were made to increase the interaction

and responses perceivable by the user. An articulated human replaced the human entity used in previous simulations. Increased ability for the human to manipulate objects was added as well as better techniques for collision detection. Sound was added to augment the natural perceptions of the environment. Alarms, bells, and 1MC announcements are just a sample of the aural effects invoked. [OBYR95]

Along with sharing a common motivation with this thesis, the O'Byrne work is important because of the introduction of an articulated human into the environment. This is essential if the VE is to be suitable for training. O'Byrne replaced the previous 3-D human model, which lacked movement and was used only as a visual aid for the user's location, with a model utilizing the *Jack*® Motion Library of the University of Pennsylvania. This addition has added immensely to the perceptual content of the damage control environment. Players can see one another interacting with the environment. One user can watch another open and close a door, or reach for objects or users can gesture to each other.

D. NOBLES and GARROVA: Shiphandling Simulator

In June 1995, Joseph Nobles and James Garrova tackled the problem of ship-handling training for junior officers. They point out that training for officers-of-the-deck suffers from the same problems as discussed previously. Desiring to offer an alternative to large-scale simulators, their objective was to implement a ship-handling trainer that could be run from a single, high-speed workstation such as a Silicon Graphics Inc. Reality Engine series model. Their result was SHIPSIM, an interactive networked real-time virtual environment for maneuvering a ship in various evolutions and suitable for shipboard installation [NOBL95].

SHIPSIM uses a split-screen configuration of a single monitor for its exercises with part of the screen dedicated to a view of the ship's surroundings and the other portion containing a graphical user interface for ship's controls. It also possesses a distributed network capability which allows multiple ships to interact within the same scenario. This

facilitates additional training evolutions, such as Underway Replenishment or DIVTACS, within the VE.

SHIPSIM also has much in common with NPSNET. Ship entities in SHIPSIM communicate in the networked configuration via the DIS protocol, as do NPSNET entities. Both SHIPSIM and NPSNET perform the same simulation loop steps, while allowing the multiprocessing of cull and draw processes. These facts made apparent that the integration of SHIPSIM and NPSNET would be inevitable [STEW96].

Though very successful in showing the ability to have low-cost, shipboard ship-handling simulators, SHIPSIM lacked in some areas, especially in taking advantage of many of the immersive qualities that virtual environments offer. These deficiencies drove future work that integrated SHIPSIM and NPSNET and included assimilation of the DC VET.

E. STEWART: Integration of Naval Training Assets into NPSNET

As previously mentioned, NPSNET is essentially an Army driven project for battlefield simulation. Coincident to much of the research implemented into NPSNET, some masters students with naval backgrounds developed independent work with a naval slant. Much of this work (some discussed above) was independent but had its basis in NPSNET. In March 1996, Bryan Stewart successfully combined the best features of the DC VET and SHIPSIM while linking the new environment into NPSNET. The new environment incorporated many features to increase the immersive feel, including sound, voice recognition of commands, and HMD capability. Perhaps the most important aspect of the Stewart research was the implementation of a capability for human entities to embark or debark ships or buildings that are themselves virtual environments [STEW96].

F. SUMMARY

Prior to the work presented in this thesis, students had developed solutions to various problems associated with training navy personnel in today's world. Each had it's

own focus but all had as their foundation the need to expand the horizons of current training methods. Their work has resulted in an extremely realistic ship virtual environment that has great features for shiphandling and damage control training. However, one factor that was sorely lacking was an emphasis on the mission of today's naval vessels - projection of power. Much of the training aboard ships is geared toward the proper and efficient operation of the many weapons systems aboard. In order to fully realize the advantages of using a virtual ship, weapons and weapons control capabilities are a must. This is the driving factor behind the research presented here.

III. SYSTEM OVERVIEW

A. THE NAVY CIC

The NPSNET Combat Information Center Virtual Environment is designed to reflect the functions of naval warship CICs. The combat information center aboard ship acts as the nerve center when the ship is in a warfighting role. It is here where tactical decisions are made and actions are taken that allow a ship to carry out her mission, be it sea control, overseas presence, sea-line-of-communication interdiction or projection of power. It provides a central location for a commanding officer or tactical action officer to evaluate the large amount of varied information necessary for proper command and control. The tasks conducted in CIC focus on two items: information and weapons.

1. Information

CIC handles information in a number of different ways, but each of the methods it uses can be grouped into one of the following four categories of information processing: gathering, display, dissemination, and evaluation. Together these categories represent the majority of the workload in the CIC. All watchstanders from bearing takers to missile engagement coordinators participate in this vital shipboard operation. In fact, barring actual hostilities, the processing of information is the single largest mission for a maritime asset. In order to successfully undertake the required tasks, modern naval ships combine technology and tradition to create realistic and accurate descriptions of both the tactical and strategic pictures.

a. Gathering

Today's warship's are equipped with a variety of tools for the gathering of information. These include equipment that can collect information actively or passively. It also includes equipment for voice and electronic communications. Some sensors, such as search radars, electronic emission exploiters or helicopters, provide a direct means by

which a ship establishes a picture of its environment without having to rely on other entities to provide it.

Surface and air search radars are perhaps the most obvious of shipboard information gathering methods (Figure 2). Perched on the highest points of the superstructure, radar enables a ship to "see" well beyond the visible horizon. In CIC, trained operators combine information provided by radars to present the best possible picture. Radar equipment comes in various types. As a minimum, an average-sized warship

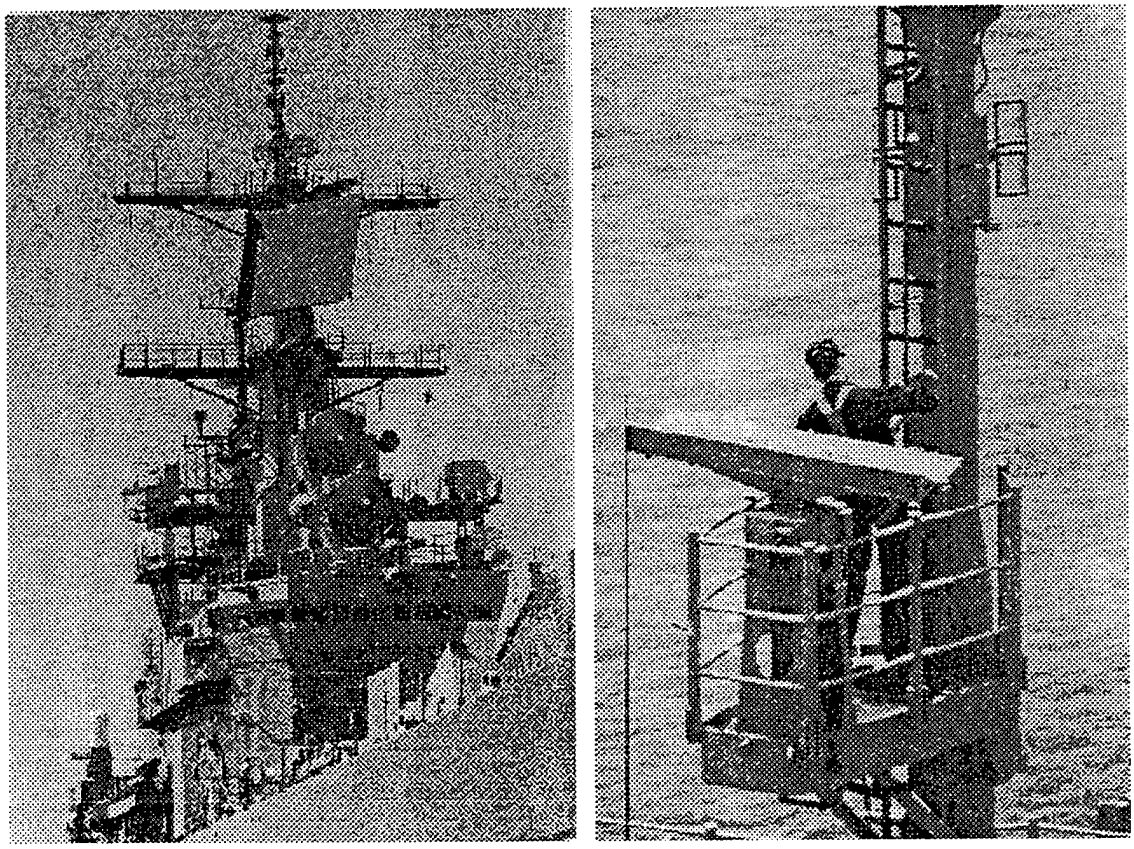


Figure 2: Air and Surface Search Radars

will have one surface search and one air search radar, with perhaps a few fire control radars for the weapons it carries. But, this is a minimum. Typically, one can find both two dimensional and three dimensional air search radars, as well as additional surface search

radars. Fire control radars are not used in search roles but in a tracking or illumination capacity suitable for controlling weapon flight and targeting.

The information gathered in CIC is not limited to that of hostile forces or other entities, such as airliners or merchant shipping. Radar information is also used for safe navigation. In fact, in cases of restricted visibility and maneuvering, CIC becomes the main navigation plot for the ship.

There is, however, a major problem inherent to using radar. Its function is dependent on the emission of radiowave energy; energy that can be intercepted. Thus, ships have the additional capability for passive gathering of information. This is accomplished a few different ways. One is the use of equipment that can exploit the electromagnetic spectrum to detect emissions without revealing own ship position. Special electronic warfare specialists occupy positions in CIC that become extremely important in controlled emission environments. Their job is to collect and correlate as much spectrum information as they can and create databases of information for future uses.

Passive collection can also be accomplished across network and communication links. Today warships share the data they assemble across links established over radio waves or through the use of satellites. Onboard computers receive this information and data personnelmen and operations specialists compile it with their console interfaces. Specially trained in link operations, some CIC personnel are solely responsible for processing data that can be regional, theater, or world-wide.

b. Display and Dissemination

Once collected the information is useless unless it is displayed and distributed to appropriate locations. In CIC, this is achieved by means as varied as grease pencil tote boards, sound-powered phones, computer-operated consoles, and large screen

CRT displays. The goal is to provide all shipboard stations with the information necessary for each station to develop a clear tactical picture.

For example, a surface warfare coordination officer in CIC would have a number of CIC personnel working for him. These would include surface trackers, link operators, bearing plotters, electronic warfare (EW) technicians, as well as the surface weapon control operators. The surface tracker and link operator collaborate in determining what surface contacts may be. The tracker will display symbology over a radar return that will identify it as friendly, hostile or unknown. This identification most likely is correlated through link information or voice communication by the link operator. The tracker is also responsible for ensuring that sufficient updates are made so that bearing, range, course, and speed for the contact are accurate. His input shows throughout the CIC on various repeaters as well as being transmitted through a link to other ships. Unique track numbers are assigned and used so that no confusion arises as to which contact is which. Meanwhile, plotters plot EW information that will aid in the identification of unknown contacts.

All this information is made available to a wide audience. Track information is displayed on weapons consoles and written on tote boards. Wall mounted large screen displays give access to the information at a glance. Repeaters on the bridge and in radar rooms allow others such as the Officer-of-the-Deck to "see" the picture develop. Once disseminated and displayed, the information must be evaluated.

c. Evaluation

The Commanding Officer or perhaps an appointed tactical officer makes the ultimate operational decisions for his ship. It is his evaluation of the information that is the most important. However, he relies on everyone aboard to conduct their own evaluations; radar trackers decide what radar returns are valid, EWs determine electronic signatures for

contacts, and Warfare Coordinators label and prioritize threats. The combined evaluations or reading of the data become the basis for the use of weapons.

2. Weapons

Unless actual hostilities have broken out, the majority of the resources of the CIC are steered toward the processing of information. However, a ship must always be prepared to act. It does so with its armament.

Like shipboard sensors, there are many different weapons available to a Commanding Officer, ranging from 0.76 caliber machine guns of the close-in-weapons-system to long range guided missiles. Centralizing control of the weapon systems is another function of the Combat Information Center. Many of the same consoles used for display of information are also weapons control consoles. This dual duty is further enhanced by the programmability of the consoles. This adds the capability for one console to be configured to interface with different information and weapons systems. Thus a tactical engagement officer could easily switch from a surface to air picture in seconds.

Continuing the above discussion of CIC operation, the Surface Warfare Coordinator, satisfied the information he has received is valid and under orders, begins the process of engagement. Together with a surface missile operator, he plans an engagement using his console to construct a salvo and flight plan. Their plans are repeated at tactical and command consoles in support of the command-by-negation concept; if those with launch approval don't approve, then plans are scrubbed and reworked.

B. DESIGN PHILOSOPHY

The CICVET was created to reflect the primary functions of a CIC and to increase the capability of naval assets within NPSNET. By implementing sensors and weapons, users within the ship VE can perform operational training and weapon exercises within the virtual battlefield.

The implementation of a functional CIC began with the framework provided in the Shipboard Simulator. This simulator was the culmination of previous work to incorporate

a Navy into the NPSNET environment. The Shipboard Simulator allows ship entities to interact with other networked land, air and sea vehicles, while at the same time allowing human entities to mount and remotely control the vehicles [STEW96]. The framework also provided the static models of a CIC interior, including displays and consoles.

C. SENSORS

1. Large Screen Displays

Within NPSNET, both human entities and ship entities inherit certain functions by virtue of the parent classes from which they were derived. While the majority of these member functions are essential for full incorporation within the environment, some produce artificial and non-realistic capabilities. For example, previously the human entity had the capability of determining positions of dynamic entities within the world through the use of a heads-up display (HUD). This HUD could be toggled on or off regardless of the human's location. The HUD uses icon symbology and fixes the icons dependent on the transmitted locations of the networked entities. While aiding in giving a sense of direction, its use as a training aid is artificial and could be detrimental for the training of anyone save pilots. A perhaps more realistic approach is to provide the location information within the world itself. In the context of a ship's CIC, the HUD data equates to a large screen display (LSD) reflecting the link information available to today's ships at sea. Presenting the information in such a way simulates the real-world links military assets share. Icon data, while still read in the same manner, takes on a connotative meaning that has an equivalent in the real-world.

2. Radar

Aside from the general direction and range information provided when the HUD is active, entities have little recourse in searching for other entities. Typically, scenarios demonstrate the use of the HUD to steer the visual location of entities. This type of search is impractical for naval use since most of a ship's engagements occur over-the-horizon and

involve long/medium range missiles or third-party-targeting. One solution would be to obtain more accurate position information on dynamic entities; information that could establish a fire control solution. This, however, ignores some real world uncertainty that is present when attacking from distant locations. Obtaining entity positions for a fire control solution from a network where updates are required for proper graphical coordination between entities would be like shooting fish in a barrel; a miss seems unlikely. Additionally, position information is only relayed for dynamic objects. Static objects present no obstacle. Thus aircraft could not fly nap-of-earth, shielding themselves with terrain, nor could ships duck behind islands or hide amongst oil derricks, as any ship driver in the Persian Gulf has experienced. With a capability to detect all objects, ship entities could navigate unfamiliar waters or simulate restricted visibility for shiphandling.

Implementing such a capability requires a look at what happens in the fleet. Since radar is the primary active way of accruing such information, installing a functioning, albeit simulation, radar was of utmost importance in the CICVET. The implementation results in data suitable for fire control or navigation and mirrors closely real world operation.

D. WEAPONS

1. Weapon Control Console

Creating a naval asset that can pose a threat in the virtual battlefield began by examining methods by which to interface with, and control, weapons. To create the most realistic VE console, users inside the CIC should be able to seat themselves at an operating weapons control console (WCC). This is opposed to simply providing a 2D interface, remote and independent of the environment. The implementation in the CICVET is a compromise, combining aspects of both methods. The compromise follows from the realization that there are limitations to the creation of a console in the world. These limitations are on the sufficient detail and interaction required to accurately simulate a WCC while maintaining adequate frame, interaction, and response rates. Elementary picking capabilities and limited articulation of human entities aside, the biggest restriction

came in the maintaining of an appropriate frame rate within the world. The memory requirements and computations necessary, when implemented in the NPSNET code, created a noticeable slowdown in frame rate which equates to detrimental effects for the user.

The result is a weapons interface that is controlled by a separate process but is initiated from within the world. This control panel communicates with the NPSNET ship and human entities through the use of a datapool. The datapool is an area of shared memory accessible to any processes that "know" the identification of the appropriate memory block. Via the datapool, NPSNET relates information about the world to the interface for display and interaction.

2. Weapons

The ability for the NPSNET ship entity to fire weapons is inherited as a derived class of entities. Both missiles and guns are available. However, the previous implementation pays no regard to the unique aspects associated with maritime weapons. These include vertical launch, waypoint maneuvering, sensor guidance, and third party targeting.

Weapons fired within NPSNET were obviously created based on air and ground targeting concepts. In effect they are fired based on visual targeting. Entities search the world for hostile forces and, using cross hairs centered on the point of view, fire weapons. These weapons fly trajectories appropriate to their type (missile/gun) and hopefully intercept the other entities position for a kill. This type of weapon control does not translate to shipboard use. In very few instances are naval weapons fired on visual information. Thus, a new functionality for NPSNET weapons had to be implemented.

Rounds fired from naval guns differ little from those in NPSNET. However, the slewing and training of the gun, and controlling the flight of the round, must be associated with information gained from other than visual means. Human entities must have the capability to point a gun along an arbitrary bearing and range. Similarly, missiles should

have the ability for coordinate launch, or flying to points in space. Without a visual input, this means that the missiles should be programmable prior to launch and, for further realism, should have a search capability of their own.

The CICVET accomplishes the above functionalities. Using the weapons control panel, both guns and missiles can be targeted using world coordinates. No entity information is used other than that of the driven entity and that returned by the simulated radar.

E. SUMMARY

The design of the Combat Information Center virtual environment parallels the capabilities found aboard today's warships, taking into account both information and weapon resources. It introduces new methods for simulating sensor technology, a radar; information display/dissemination, LSDs; and weapon interface, the WCC. The VE is built upon the NPSNET framework and has its foundations in the previous research that resulted in a naval entity within the virtual battlefield.

IV. DEVELOPMENT TOOLS

A. OPENGL

"OpenGL is a software interface to graphics hardware" [OPEN93]. OpenGL is a graphic library developed by Silicon Graphics, Inc.(SGI) with the open architecture idea in mind. In other words, it is a hardware independent API which senses the current architecture and translates the software calls to the instructions of existing hardware. OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms. To achieve these qualities, no commands for performing windowing tasks or obtaining user input are included. It can perform all of the tasks in the software level, or leave some of them to the hardware, depending on the existing hardware. "OpenGL doesn't provide high-level commands for describing models of three-dimensional objects. With OpenGL, you must build up your desired model from a small set of *geometric primitives* - points, lines, and polygons" [OPEN93]. It has features like wireframe modeling, depth-cueing, antialiasing, flat and smooth shading, texture mapping, and lighting features. "OpenGL is a state machine. You put it into various states (or modes) that then remain in effect until you change them" [OPEN93]. OpenGL can be used in the following areas of interest;

1. Using mathematical expressions to build shapes from geometric primitives like lines, points, and polygons.

2. Mapping out objects in a three-dimensional scene and use the desired camera point to view the arranged geometric scene.

3. Computing the color of objects dynamically depending on the environmental conditions.

4. Displaying those mathematical expressions and colors on the screen by translating them into pixel information. This final and one of the most difficult jobs is named *rasterization* [OPEN93].

B. IRIS PERFORMER

1. Description

A product of SGI, IRIS Performer is a software toolkit for the development of real-time visual simulation and interactive graphic applications. Supporting the industry standard graphics library, OpenGL, Performer allows the creation of applications suited for optimal performance on SGI graphics hardware.

2. Features

a.Hierarchical database

A visual database, also known as a scene, contains state information and geometry. A scene is organized into a hierarchical structure, specifically as a directed-acyclic graph as shown in (Figure 3).The scene hierarchy supplies definitions of how items in the database relate to one another. Additionally logical and spatial organization information of the database is also held within the same hierarchy. The scene hierarchy is traversed by visiting the nodes in depth-first order and operating on them. "IRIS Performer implements several types of database traversals, including application, clone, cull, delete, draw, flatten, and intersect" [IRIS95].

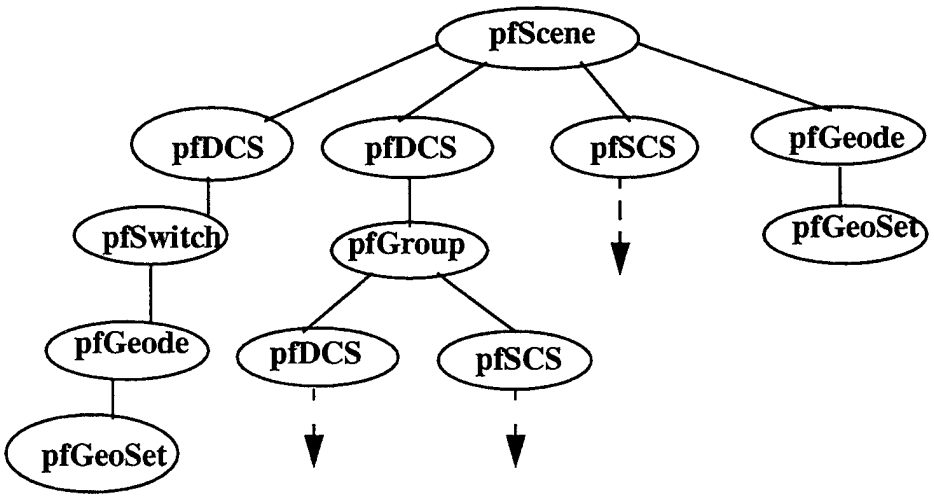


Figure 3: Database hierarchi of a Performer scene

The root of a visual database *pfScene* is viewed by a *pfChannel*, which is rendered to a *pfPipeWindow* by a *pfPipe*. Process flow for a single-pipe system is shown in (Figure 4). The application constructs and modifies the scene definition (*pfScene*) associated with a channel. The traversal process associated with that channels's *pfPipe* traverses the scene graph, building an IRIS Performer *libpr* display list. As shown in the (Figure 4), this display list is used as input to the draw process that performs the actual graphics library action required to draw the image [IRIS95]. Any number of channel and pipe combinations are possible in IRIS Performer, however a minimum of one of each type is required.

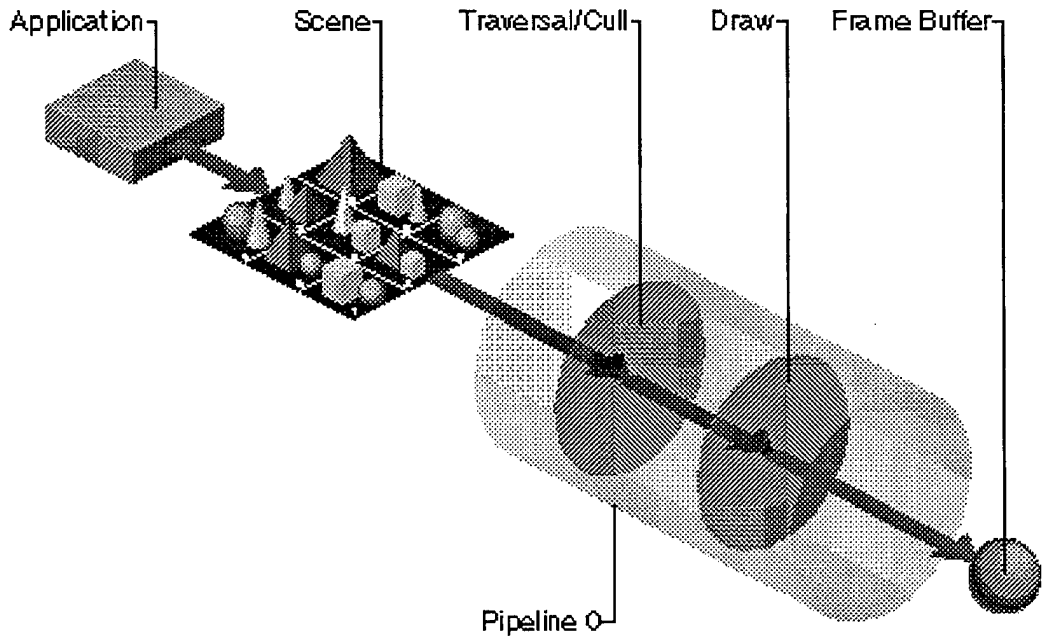


Figure 4: Single Graphics Pipeline [IRIS95]

b.Libraries

IRIS Performer consists of basically four main libraries. *Libpf* is the main library which handles multiprocessed database traversal and rendering and also contains *libpr* which performs the optimized rendering, state control, and other functions

fundamental to real-time graphics(Figure 5). *Libpfdu* is the library of scene geometry building tools which greatly facilitate the construction of database loaders and converters. Tools include a sophisticated triangle mesher and state sharing for high performance databases. *Libpfutil* and *libpfui* are utility-functions and user interface libraries respectively [IRIS95].

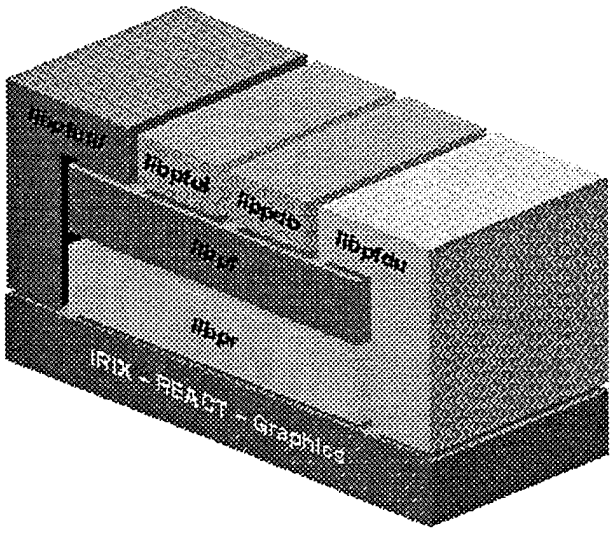


Figure 5: Performer libraries [IRIS95]

c.Multiprocessing

One of the most important features of IRIS Performer is its ability to use existing multiprocessor architecture to speed up real-time graphics applications. By dividing complex processes into small pieces it allows the processes to execute in parallel. "IRIS Performer uses multiprocessing to increase throughput for both rendering and intersection detection. Multiprocessing can also be used for tasks that run asynchronously from the main application like database management" [IRIS95]. Shared memory management is a big problem in multiprocessing. IRIS Performer partitioned those problems into three categories, as their solution. These categories are memory visibility, memory exclusion, and memory synchronization.

3. Execution

Execution of a IRIS Performer scene follows certain steps in the process pipeline. Initialization and configuration of IRIS Performer is followed by setting up a pipe and a channel. Frame rate and synchronization, creating and loading the scene graph, and the simulation loop are the follow on steps of the initialization.

a. Initialization and Configuration

Initialization consists of two main levels, shared memory and multiprocessor initialization. Performer function *pfInit()* sets up the shared memory arena which is being used to share data among the applications, the visibility cull traversal, and the draw traversal, all of which can run in parallel on different processors. Another procedure *pfConfig()* starts up multiple processes, which allows visibility culling and drawing to run in parallel with the application process [IRIS95].

b. Setting Up a Pipe and a Channel

A pfPipe variable represents an IRIS Performer software graphics pipeline. In IRIS Performer each rendering pipeline draws into one or more windows associated with a single Geometry. A channel is a rendering viewport into a pipe. A pipe can have many channels in it, but by default a channel occupies the full window of a pipe [IRIS95].

c. Frame Rate and Synchronization

The frame rate is the number of times per second the application intends to draw the scene. The period for a frame must be an integer multiple of the video vertical retrace period, which is typically 1/60th of a second. The synchronization mode or phase defines how the system behaves if drawing takes more than the requested time. *pfSync()* delays the application until the next appropriate frame boundary [IRIS95].

d. Creating and Loading the Scene Graph

Database representations of models can exist in a variety of formats. IRIS Performer doesn't define a file format for a database; instead it supports extensible run-time scene definitions of sufficient generality to support many database formats [IRIS95]. You can create a database with any modeler then import it into a Performer view. Additionally when it comes to load the database, Performer keeps the original database hierarchy inside its own view.

e. Simulation Loop

After the pipes and channels are configured and the scene is loaded, the main simulation loop begins and manages scene updates, viewpoint updates, scene intersection inquiries, and image generation [IRIS95]. It has two main control functions; *pfSync()* to put the process to sleep until the next frame boundary, and *pfFrame()* to initiate the next cull traversal. Time consuming calculations are best placed after *pfFrame()* but before *pfSync()* while other small calculations can be executed in the remaining half cycle.

C. MULTIGEN

1. Description

MultiGen is a three dimensional graphics editor that lets you create and modify visual system database in an intuitive "what you see is what you get" environment. Multigen has been designed to support many different visual system formats [MULT95].

2. Features

MultiGen is used to create, edit and view 3D scenes (databases) used in visual simulation. It allows a scene to be viewed as it will appear on the image generator, including Silicon Graphics workstations. Unlike general purpose modeling tools, MultiGen is optimized to handle level of detail switching, instancing, hierarchy, and external model references. Modelers can easily create and edit 3D objects, database hierarchies and

attributes by direct geographical manipulation. Geometry, color, texture, shading, and application-specific attribute information is stored in the MultiGen database for interpretation by the real-time application code.

D. RAPIDAPP

1. Description

RapidApp is an application builder, a component of the Developer Magic Application Development Environment for developing applications to run on Silicon Graphics workstations. This integrated development environment provides tools for rapid application development. RapidApp uses IRIS IM container widgets, a collection of containers that provide a variety of ways to organize interface elements. To maintain their organizational features, many containers keep a tight control over their child elements, often determining their size and position by the order in which the elements were created.

2. Features

RapidApp is a graphical tool which allows developers to interactively design the user interface portion of their application. It generates C++ code utilizing IRIS ViewKit for each user interface component as well as the overall application framework. As with all ViewKit-based applications, IRIS IM (Motif) widgets are used at the bottom level as the basic building blocks for the user interface. To speed the development cycle RapidApp is integrated with a number of the Developer Magic tools, including: cvd, cvstatic, cvbuild, Delta C++, and Smart Build. This allows developers to quickly design, compile, and test object-oriented applications. Another benefit of RapidApp is the built-in support for producing an application ready to integrate with the IndigoMagic Desktop Environment. This involves automatically generating things like an ftr file for file-typing-rules, an icon for the desktop, inst image information, etc.

Finally, RapidApp also provides easy access to using SGI-specific widgets and components within an application. For instance, an OpenGL widget can be added to a

program without having to know many of the underlying details of integrating OpenGL and X.

V. LSD IMPLEMENTATION

A. SEARCHING MODEL TREES

With little or no knowledge of how the ship model was put together, we began by familiarizing ourselves with Multigen, the model software used in previous theses to construct the interior spaces of the Antares ship model. Both Multigen and Performer arrange a model's data in a tree structure with actual geometry contained in leaf nodes. Both also allow the naming of nodes for easy reference. By comparing the tree of the ship model in Multigen with that of the one generated in Performer (we accomplished this by using the "Show Tree" function of the Performer example program Perfly) we were able to determine that Performer maintained the same names used in Multigen for all nodes except for those containing actual geometry (or the equivalent of a pfGeode in Performer). A Performer programming attempt to search for the actual polygon was unsuccessful verifying that, indeed, the Multigen name for this node was not maintained when loaded by Performer calls. Our hope had been that all names would be carried over into Performer, allowing for the simple traversal of the model tree in a search for the name of the polygon representing a particular LSD. This not being the case, we were only able to search using node name down to two levels above the actual geometry. The "found" node was the equivalent of a Performer pfGroup node, used to group together nodes in logical clusters. Once we had access, via pointer, to the CIC and LSD group nodes and not being able to proceed further down the tree by name, we proceeded by child number. Each pfGroup node has a member function returning the number of children. Being indexed sequentially, a particular child can be returned if its index is known. Retrieving the index for the LSD polygon from the Multigen tree, we were able to access the LSD geometry node.

B. INFORMATION DISPLAY

The next step was to figure out how to actually display information on the LSD. As discussed previously, NPSNET already produced location information on dynamic entities

and displayed it in the form of a HUD. This same information drives the drawing of the LSD.

1. Locating new geometry

After finding the correct node that houses the LSD model geometry, the trick was how to and where to locate the new geometry. Performer does not provide facilities for easily recovering the coordinates for particular geometry sets. Each GeoSet can return pointers to the blocks of memory that contain the coordinates, but the order of the coordinates, number of vertices, primitive type, and number of primitives must be queried separately. And, while Performer does not allow for direct dynamic change of vertex coordinates, entire GeoSets can be replaced with others. Once we identified the correct vertex coordinates, we used normal calculations to determine proper placement of the new geometry.

Once the coordinates are found, we calculate the surface normal, the center, and the radius of the biggest circle that can fit in LSD. Figure 6 demonstrates the method used to retrieve the four corner coordinates of an LSD and the calculation of the surface normal. The surface normal is computed by determining two vectors that lie on the plane of the LSD. These vectors (variables first and second in Figure 6) are at right angles to one another and are of unit length. The normal, representing the orientation in space of the LSD, is the cross product of the two planar vectors.

```
 1 pfGetGSetAttrLists(geo, PFGS_COORD3, (void**)&coords, &icoords);
 2
 3 pfVec3 first;
 4 pfSetVec3(first, (coords[0][0] - coords[1][0]),
 5                  (coords[0][1] - coords[1][1]),
 6                  (coords[0][2] - coords[1][2]));
 7 pfNormalizeVec3(first);
 8 pfVec3 second;
 9 pfSetVec3(second, (coords[0][0] - coords[3][0]),
10                    (coords[0][1] - coords[3][1]),
11                    (coords[0][2] - coords[3][2]));
12 pfNormalizeVec3(second);
13 pfCrossVec3(norms, first, second);
```

Figure 6: Sample code describing surface normal calculations

Figure 7 details the calculation of the LSD center and the determination of the radius of the largest circle that will fit within the surface. Along with the normal, which describes orientation, the center, in world coordinates, gives the absolute position about which other objects can be located.

```
1 //Calculate the center of LSD
2 temp1[0] = (coords[0][0] + coords[1][0]) / 2.0;
3 temp1[1] = (coords[0][1] + coords[1][1]) / 2.0;
4 temp1[2] = (coords[0][2] + coords[1][2]) / 2.0;
5 temp2[0] = (coords[3][0] + coords[2][0]) / 2.0;
6 temp2[1] = (coords[3][1] + coords[2][1]) / 2.0;
7 temp2[2] = (coords[3][2] + coords[2][2]) / 2.0;
8 center[0] = (temp1[0] + temp2[0]) / 2.0 + norms[0] * 0.01;
9 center[1] = (temp1[1] + temp2[1]) / 2.0 + norms[1] * 0.01;
10 center[2] = (temp1[2] + temp2[2]) / 2.0 + norms[2] * 0.01;
11
12 //Calculate the radius of LSD
13 radius = sqrt(((temp1[0] - center[0]) * (temp1[0] - center[0])) +
14          ((temp1[1] - center[1]) * (temp1[1] - center[1])) +
15          ((temp1[2] - center[2]) * (temp1[2] - center[2])));
```

Figure 7: Sample code demonstrating center and radius calculation

2. Range rings

The first geometric objects added to the LSD were rings representing ranges from the ship. The rings were constructed using Performer geometry building functions. The default object was a unit circle at position (0.0, 0.0, 0.0) with normal vector (0.0, 0.0, 1.0). To co-locate the rings with the LSD we translate it to the center of the LSD, scale it by the radius calculated in the previous step, then rotate it. We need to calculate the rotation axis and angle. The cross product of the normals (unit circle and LSD) gives the rotation axis, while the arcsin of length of cross product gives the rotation angle. After the rotation, we translate the rings a little bit further in the direction of LSD's normal to prevent "flimmering" or the effects of z-buffering and co-planar geometry. Finally, the ring geometry is added to the parent node of the LSD. This ensures that movements of the ship will affect the rings, leaving them in the same relative location. The ranges the rings represent are presented as labels on the LSD.

3. Icons

a.Symbology

The HUD employed by other entities in NPSNET uses special icons to represent not only approximate range and direction but also to indicate what type the remote entity is. The icons are drawn from a special bitmap font. Since the HUD is constructed from direct OpenGL calls, using a 2D orthographic projection, the icons can simply be output as a text string at window coordinates that equate to a relative position in the world. Using a similar methodology for the LSD is impossible since there are no facilities for drawing text in three-space. Performer does offer a special node, pfFont, for three dimensional text, but current implementation (Performer 2.0) has only two native fonts available and no facility for importing bitmap fonts.

The solution was to build icons as 3D objects suitable for import into a Performer hierarchy. The icons exist as individual files that are read during the loading of models. The models are not attached to the scene but rather exist in an array structure that is indexed in the same way the icon fonts are in the HUD code. Whenever an entity enters the world a new icon is attached to the scene and its geometry is created by copying the appropriate symbol from the icon model array. Matching the array indices to the same ones used by the HUD enables the reuse of code. No new determinations of entity type need be made. If an entity drops out its icon is removed from the scene and deleted, returning the memory it used to the heap.

b.Positioning

Situating the icons on the LSD happens in much the same way as the rings were located. A linked list data structure is used to maintain information on the dynamic objects interacting within the virtual world. Each time through the main simulation loop a check is made to determine if any entities have joined or left the simulation. If so,

appropriate alterations are made to the linked list. During the update entity stage, this list is read and position information is obtained for each entity. Since the information is in true world coordinates a conversion is required to make a position relative to the ship's location. The range is ratioed to fit within the area of the LSD.

C. SUMMARY

The large screen display is used to represent information on other entities participating within the same simulation. An argument can be made that absolute knowledge of the whereabouts of other entities is unrealistic and contrary to any real world functionalities. However, today assets can share information in real-time. The LSD information can be viewed as information obtained from satellite or electromagnetic link-ups. In fact, simple changes to the algorithm determining which vehicles to display can be made to more genuinely reflect operations. Vehicles could be entered only by entities that have actual knowledge of its existence, i.e. visual or radar information.

Locating specific geometry and obtaining its coordinates is essential to create a dynamic environment representative of real-life CICs. The ability to manipulate objects on the fly is required to properly display the information collected. Using vector notation for defining planes and normals for orientation, new geometry can be co-located with existing objects.

VI. RADAR DEVELOPMENT

A. MOTIVATION

Aboard ship, radar plays a major role in the gathering of information. To properly replicate a CIC working environment, a suitable rendition of radar needed to be implemented. Chapter III touches upon some of the differences in the information generated by previous NPSNET versions and that required by ships.

There are two main reasons for creating the CICVET radar. The first is that simulating a radar adds some uncertainty to the information. No longer are exact entity positions used. Now, in the absence of link information, only if the ship entity itself detects an object will it be able to see it. In fact, entities can exploit this fact to hide their position and conduct clandestine operations. This will add a qualitative element to the training of tracking operators and radar users.

The second justification for radar in NPSNET is that it allows for knowledge of static objects within the world. Because the method used to simulate the radar makes no distinction different between dynamic and static objects, the radar "paints" images of everything it hits. In particular, this functionality works well for land masses, enabling navigation by other than visual means.

B. METHOD

1. Native Performer Collision Detection

The radar is built upon the intersection functions provided by IRIS Performer's libraries. Performer's collision detection is centered around a structure called a pfSegSet. This data record houses information needed to construct groups of line segments. Up to 32 of these line segments can be defined within a single pfSegSet. During an intersection traversal, these line segments are compared to the scene database and collision information is returned.

Within a pfSegSet, two bit masks exist to further delineate information provided during the intersection traversal. The active mask is used to individually turn on and off line segments. The isect mask is used to discriminate intersections. As a binary number it can be set to selectively decide which objects a traversal will intersect with. All objects within a scene also have their own intersection mask. Two main tests are conducted during intersections. The first is physical, answering the question "Does this line segment pass through an element?" The element can be set to a geometry's bounding volumes or to its actual primitives. The second test is for discrimination, answering, "Is this intersection of the type requested?" The discrimination is carried out by a bitwise AND of the traversal and object intersection masks. If the boolean AND results in something other than all zeros, the intersection information is saved for use in the application process.

The structure used to save this information is a pfHIT in Performer. After a successful intersection, these objects hold data including: the coordinates of the intersection, the normal vector at the point of intersection, the transformation matrix of the intersected object (used for converting intersection points to world coordinates), and identification information of the intersected node. By default, only one, the first, intersection is determined. This, however, is easily changed with traversal modes and callback functions. Extracting the information entails submitting queries to the pfHit structure, which in turn replies with the appropriate data response.

2. Intersections for Radar Simulation

The design for the CICVET radar encapsulates many of the operating characteristics of shipboard radar, but at the same time it is in no way physically modeled. The strategy for radar implementation was driven by the desire to approximate the information radar gives in an effort to augment NPSNET's capabilities. While successful, the radar has limitations. Radar enhancements such as physical modeling is left for future work.

a.Intersection test description

Numerous design methods were examined before the final construction was decided upon. These methods were tested first as stand alone programs, in an effort to demonstrate their effectiveness. Then the code for each was inserted into NPSNET for evaluation of their performance. One goal when creating the radar was to impact the frame rate as little as possible. Thus, performance between the various methods was crudely determined using an approximation of frame rate. A radar sweep was placed within the scene whose azimuth angle was incremented for each pass of the draw process. This movement gives an idea of the speed of the NPSNET process. Comparing the sweep cycle to the clock estimated the change in frame rate between methods.

The basic construction, upon which various radars were built, consists of a pfSegSet with a single line segment. Using a relative polar coordinate system, the line segment had one endpoint anchored at the ship's position with an initial azimuth of 0^{o} and an elevation of 0^{o}. The length of the segment was equivalent to 30 nautical miles (nm) in virtual world distance. This distance of 30nm is comparable to that of a nominal surface search radar found aboard ship. Each time through the draw process the line segment's direction would be changed, sweeping through each degree of azimuth. As the segment's direction changed the entire scene graph was traversed to determine which, if any, node geometry was intersected. The intersection points returned were placed in an area of shared memory for use in the display process. Chapter VII discusses this procedure.

The pfSegSet has an intersection mask equivalent to 32 ones or 0xFFFFFFFF in hexadecimal notation. With one exception, this enables the intersection discrimination test to be successful with all geometry when the bitwise AND is applied. Only if the geometry mask is all zeros will the discrimination fail. Also, within the pfSegSet

is a pointer to a callback function. For each intersection a line segment returns, the callback function is invoked. This enables greater discrimination capabilities.

For instance, within the callback, a test is made that throws out any points within a particular distance (in the ship's case 300 meters). This ensures that no intersection points are saved that result from intersecting the geometry of the vehicle employing the radar. Another solution may have been to set the two intersection masks (the pfSeg's and the ship's) in such a way that the discrimination test failed. This idea, however, was discarded in favor of the more general first solution which allows for possible implementation of the radar on vehicles other than a ship.

b.Radar along a single bearing

Initially, the radar was created as one would find it in the real world. The line segment was positioned at a mast head location and tilted toward the ground. Then the angle of declination was decremented along a particular bearing. Figure 8 simplifies how this would work. The numbers represent the changing declination along a bearing. Though this shows only ten, a greater number would be required to achieve a fidelity adequate

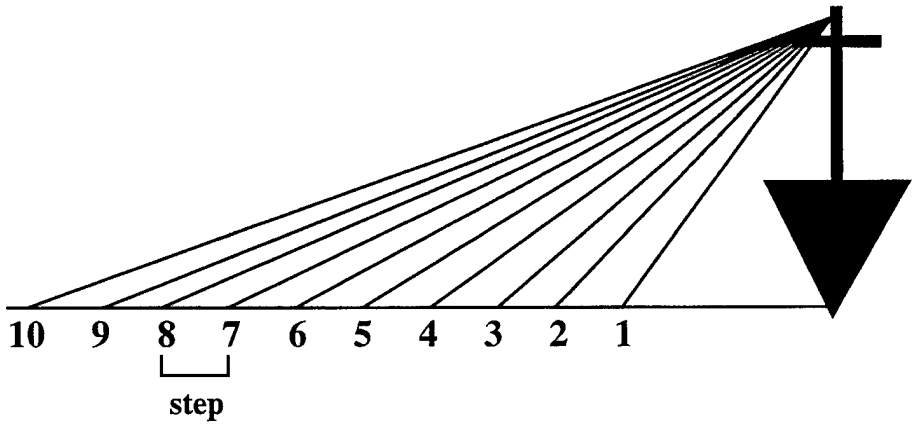


Figure 8: Mast-mounted Radar

for searching. For instance, to achieve a step size (see Figure 8) of ten meters (arguably minimal for searching) while covering 30nm (approx. 55560 meters) means 5556 steps. This is quite a large number for a control structure such as a 'for loop' when degradation in frame rate is at stake. Even when using the maximum number of segments allowed per pfSegSet there are still roughly 175 steps required to cover the entire bearing out to 30nm. Another factor contributing to the disadvantages of this method is that all objects along a particular bearing would create a radar return disregarding any shielding effects of objects in front of one another. This is somewhat unrealistic. Though real-world radar has lobing effects and a beamwidth that allows some "seeing" past an object, for the most part once an object is hit the remainder of that bearing is obscured. With these ideas in mind, a second design, represented in Figure 9, was utilized.

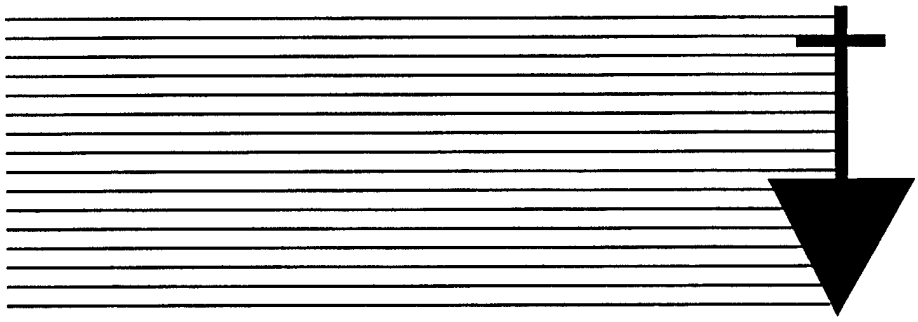


Figure 9: CICVET Implementation

The advantage of this composition is that no step size along the bearing is required, since the line segment extends along the full 30nm distance. This design, however, removes any chance of locating low-flying aircraft. To remedy this, for each bearing, the line segment was cycled through an elevation of 100 meters. The step size was

fixed to every two meters. This means that within 100m vertical distance only objects of less than two meters in height and flying directly between elevation lines would not be picked up by the radar.

c.Multi-bearing sweeps

With the above radar characteristics in place, the pfSegSet was swept through each degree of bearing around the ship's position. This produced a good quality picture of the objects in the world. However, it also had too large a gap at the farthest reaches of the radar. As can be seen in Table 1 column two, which lists the gaps at various ranges, this method is sorely lacking in accuracy of searching. For instance, at 30 nm an

Table1: Radar Gap Measurements

Range in nautical miles	Approx. Gap in meters (single line segment)	Approx. Gap in meters (between line segments)
30	969	30
25	808	25
20	646	20
15	485	15
10	323	10
5	162	5

object could conceivably be one kilometer across and not be picked up by the radar. Blind spots this large were unacceptable. An alteration was made using all 32 line segments available within the pfSegSet. By dividing the one degree angular spread between bearings into 32 pieces, an accuracy was achieved in meters roughly equivalent to the range in miles See Table 1, column three).

This incarnation produced a fidelity for accurate display of static objects such a land masses and for locating dynamic objects such as ships. At this point two obstacles still needed to be overcome. The first was that, though the radar produced satisfactory results, the sweep time (in this case the time the graphical display takes to sweep 360^{o}) was much too slow in comparison to real world surface search radars. As an example, the SPS-55 radar, found on many U.S. warships has a sweep of 16 revolutions per minute. This means that an operator would not get a realistic feel from this display because the sweep would be much slower in the CICVET than at an actual radar repeater. The contributing factors to this slow down were not only the computation time for the intersection tests but also the communication time between the intersection process and the process displaying the returns. These processes are discussed in Chapter VII.

The second stumbling block was that dynamic objects smaller than ship size (such as helicopters) could still be missed with current gaps between line segments. The solutions to both these seem diametrically opposed. Increasing sweep time lowers the possible accuracy while the reverse is true when trying to ensure smaller objects are found. This inverse relationship was avoided by developing a method that increased sweep time while decreasing accuracy only on the static objects within the world. A separate solution ensuring the location of dynamic objects, those of most importance, was implemented.

3. Radar Interlace and Dynamic Object Search

To increase the sweep times, the radar was interlaced. This entailed changing the step between bearings from 1^{o} to 10^{o}. After a completion of one full revolution, the next sweep would begin 1^{o} from where the previous sweep began. As an illustration, consider a sweep beginning at azimuth 000^{o}. The sweep would proceed every 10^{o}, thus bearings 010^{o}, 020^{o}, 030^{o},... 350^{o} would be covered for this sweep. The next revolution would begin at 001^{o}, thus covering bearings 011^{o}, 021^{o}, 031^{o},... 351^{o}. Sweeps would proceed in this

fashion until every bearing was checked. In this manner, the entire 360^{o} would be checked every 10 sweeps.

As a result, graphically, the sweep of the radar is more true to life. Even though each sweep of the CICVET radar produces 1/10th the data a real radar produces, this doesn't pose a problem within the VE when considering only static objects. These static objects under consideration are essentially land masses whose size in relation to the ship are significantly large enough that a hit every ten degrees is sufficient for determining and denoting its position. Additionally, if returns from previous sweeps are allowed to remain visible throughout the ten degree interlace then after the first ten sweeps a full picture of the world will be presented and refreshed as a appropriate. When considering dynamic objects this method would be unacceptable because of the large gaps in coverage. Dynamic objects are therefore handled differently.

In order to ensure that dynamic objects are located, the world positions of the entities are cached each time through the draw process. A calculation is made to determine the bearing of the entity from the ship. For each bearing hit during a sweep a comparison is made to determine if any entities bearing is within a 10^{o} sector of the current bearing. If so a special intersection test is made along the entities bearing. This is conducted, as opposed to simply placing a return at the entities location, to simulate the shielding effect. For instance, if a ship is located on the other side of an island we don't want it to have a return on the radar.

C. SUMMARY

Simulating a functional radar within a virtual battlefield environment is essential, especially when incorporating maritime assets, as this is a primary method of information gathering. Additionally, many of the aspects of radars need to be incorporated to provide realistic simulation. CICVET uses native Performer intersection methods to implement the radar. The radar uses two different methodologies to deliver adequate performance in the

instances of static and dynamic object searching. The first, an interlaced bearing sweep, and the second, a dedicated sweep on dynamic object bearings.

VII. WEAPONS CONTROL CONSOLE

A. BACKGROUND

With the large screen displays and driving elements of the radar in place, attention was turned toward the creation of shipboard weapons. The first step was to determine an appropriate interface by which an operator could both track contacts, determine weapon plans, and release weapons. Chapter III discusses the motivation behind the design philosophy of the weapons control console (WCC), in particular why the console was created as a separate process. Care was taken to incorporate many of the functions of WCCs found aboard ship. Figure 10 is an actual screen shot of the WCC utilized in CIC VET. The WCC can be broken into four major groups: the radar display (center), missile control (left), gun control (upper right), and own ship information (lower right).

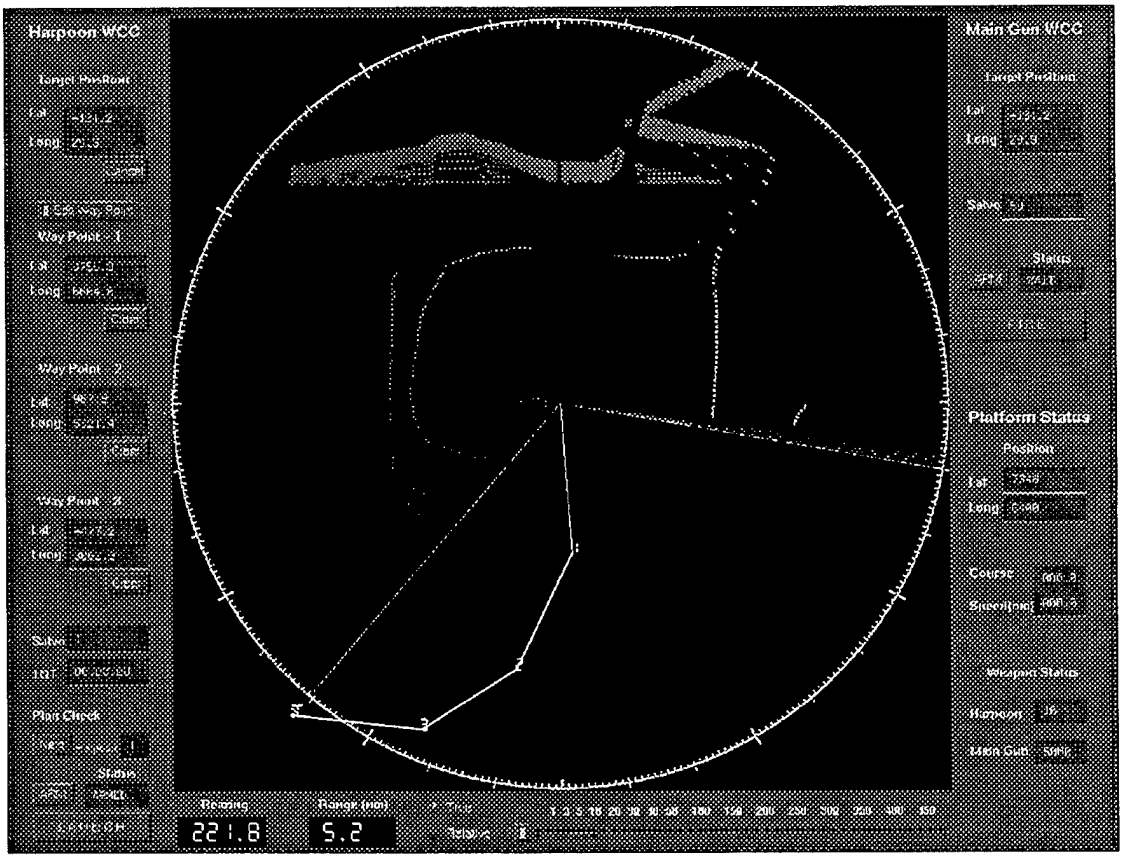


Figure 10: Weapons Control Console

The window and its widgets were created using RapidApp. RapidApp provides an easy method for creating a window and control buttons. Having more diverse widgets available than Performer, it provided more options by which to add functionality. RapidApp also builds the C++ code framework which drives the window. Functions simply need to be filled in with the code necessary to get a particular widget to act as you want it.

B. RADAR SCOPE

1. Draw Window

The center portion of the interface contains a large draw window. This window enables display of graphics created with the OpenGL architecture. It is the window by which radar returns from NPSNET are displayed. Setup as an orthographic projection, the display flattens three dimensional coordinates to two dimensions along the Z axis. Also because of the difference in the default orientations of OpenGL and Performer (See Figure 11), porting coordinates directly from Performer into an OpenGL application results in a top-down view of the Performer world by the OpenGL program. This is the desirable view when talking about radar return display. For a 2D surface search radar there is no elevation information provided, thus this setup is ideal.

2. Data Points

a.pfDataPool

The radar returns to be displayed must come from the radar implemented within NPSNET. Chapter VI discusses the particulars of how this works. Because this interface was built as a separate process, some form of communication had to be built in. We decided to do this in shared memory. Performer provides a class known as a pfDataPool for shared memory between processes. Objects of this class can contain any number of data structures along with a unique identifying string. Any process can acquire the information

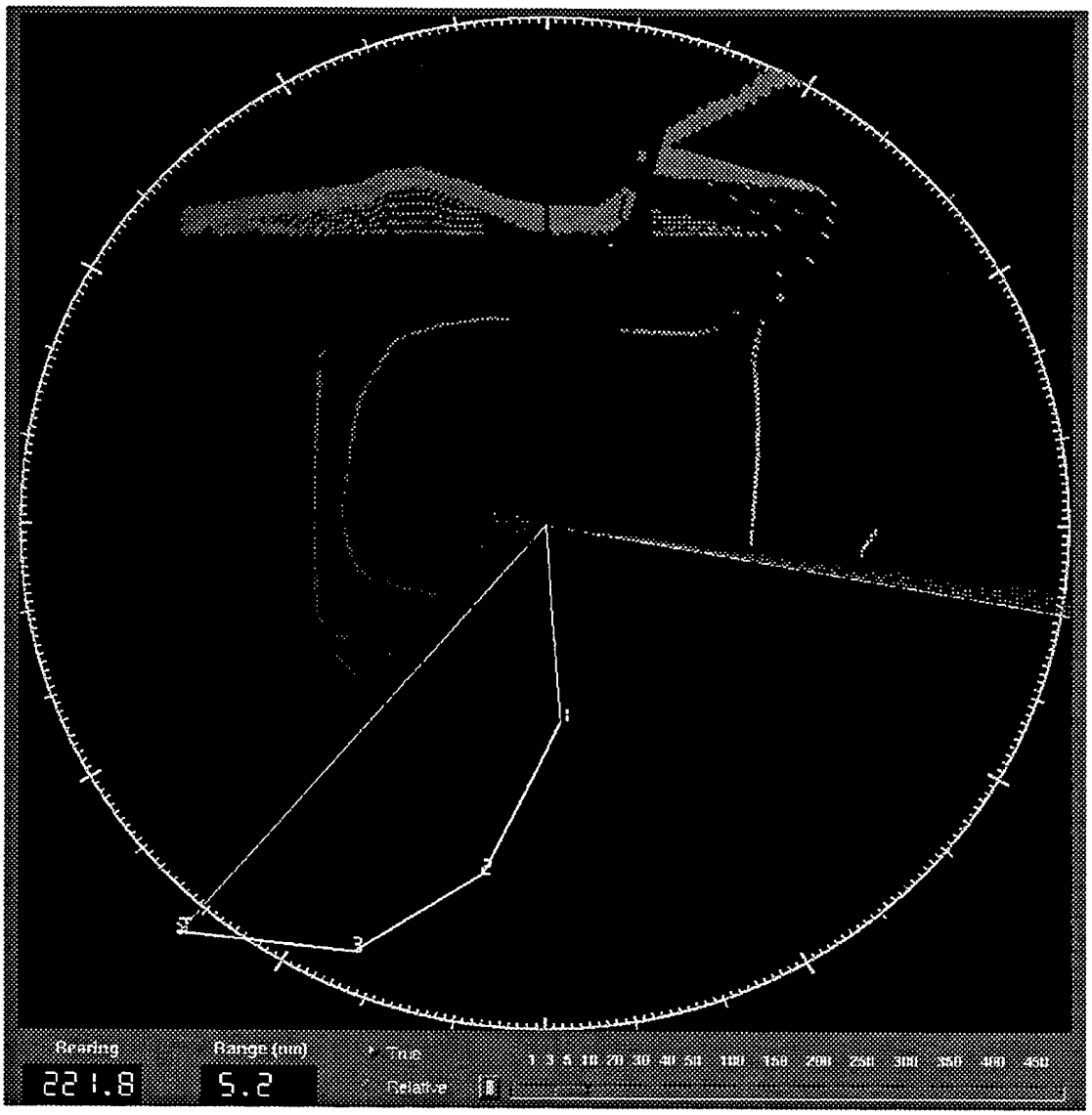


Figure 11: Radar Repeater

within the data pool as long as it knows the identifier. The memory location information is written as a file into a default directory with a name equivalent to the identifying string. Performer processes making an acquire simply read the location information from the file and access that area of memory.

This pfDataPool class also contains member functions implementing a locking mechanism. Thus the collision of processes accessing the data simultaneously is avoided. Without possession of the lock, a process is blocked and waits until the lock is free and it can proceed. Working this way protects the integrity of the shared information.

Specifically, the datapool contains a number of different variables, the largest of which is a 360 X 1600 two dimensional array of points. The 360 represents one dimension for each azimuth the radar sweeps. The second dimension size, 1600, represents the maximum possible number of points that could be returned. Each bearing is actually thirty-two line segments covering the 1^{0} angular area. Because each of these is raised in elevation for 50 steps each bearing, the maximum points possible is 32 X 50 or 1600.

b.Local storage

Within the interface program, the data points are read from NPSNET by way of the shared memory area. These points are then used by a draw function which places green 'paints' at the appropriate location. The points placed into the pfDataPool, however, are associated with one line of bearing, the particular one NPSNET radar intersected on. To simulate a radar repeater many bearings of information must be retained.

This is accomplished through the use of a dynamically linked list, where each node represents a point. Upon each read of the shared memory area, a new node is created and added to the end of the linked list. A field in the node's structure contains a float representing the age of the point. This age field is used to weight the color that the point should be drawn, eventually leading to black (no color), and thus it's removal from the list and the return of its memory. After the shared area has been read the newly appended list is passed into the draw process for display in the glDraw window.

3. Range Rings and Cursor

To complete the radar display, two more functions were added for range rings and a mouse enabled cursor. Range rings are fixed at intervals to give the operator a quick glance at far a radar contact may be. For closer scrutiny a mouse enabled cursor can be used to place a cursor over the contact. The mouse position is read and converted to world coordinates and the true (or relative) bearing and range are displayed in the lower left of the radar scope.

C. MISSILE ENGAGEMENT SECTION

1. Panel

Figure 10 is a screen shot of the missile engagement section. This section is used for display and input of missile flight profiles. For CICVET purposes the NPSNET ship was provided a capability for launch of 16 surface-to-surface missiles. The status of each missile can be reflected in the missile engagement section.

Four display windows contain latitude and longitude. One for selected target position and one each for flight waypoints. A toggle button can be set to enable input of waypoints. Although any of these windows can be accessed directly, the primary method of entry is the mouse.

Inside the radar scope, the mouse button can be used to place/edit waypoints or select targets. Once entered, a path connecting the waypoints to the target is generated. This enables a visual of the intended flight path in order to determine safe paths or to perform feint maneuvers (i.e. off bearing shots). The waypoint edits apply to whatever missile cell is currently selected. This cell is displayed as a number from 1 to 16. If at anytime a waypoint needs to be deleted the CLEAR button for that waypoint can be picked. No waypoint or target information is passed to the missile until the ARM button is depressed. This will change the status from READY to ARMED and store the flight information. The Cancel button can be used to return a cell to the READY condition.

Multiple missiles can receive the same flight paths simply by selecting new missile cells after a successful arming. Flight information is retained until it is edited for a new target or a change in waypoint. Each entry of a different missile cell increases the salvo number for that particular launch. Thus, one launch could contain any number of missiles (up to 16) on any number of different or the same flight paths to various targets. A much more desirable situation for naval assets than line of sight engagements.

2. Missile Data Passing

The structure of the missile flight information resembles that of the radar point data. Arming a particular missile cell stores the flight path in a data structure. This data structure relays the information to NPSNET via shared memory, again, in the form of a pfDataPool. NPSNET uses this information to dynamically construct the required missile objects (discussed in Chapter VIII) and awaits launch notification. At launch, these objects are released as entities with the NPSNET environment.

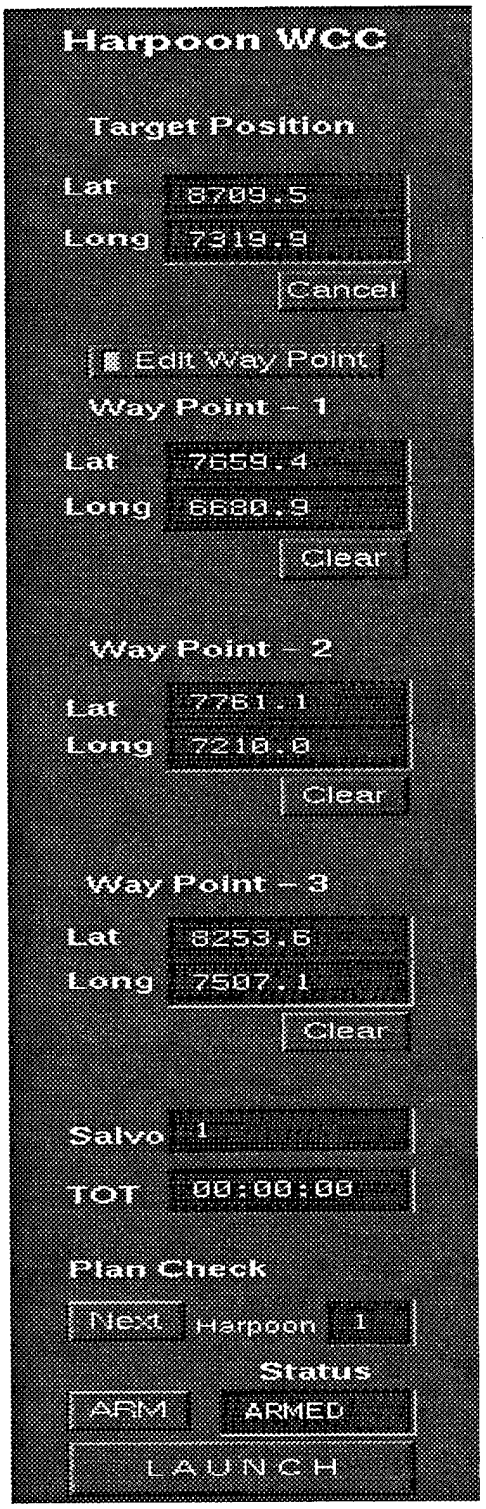


Figure 12: Missile Engage Section

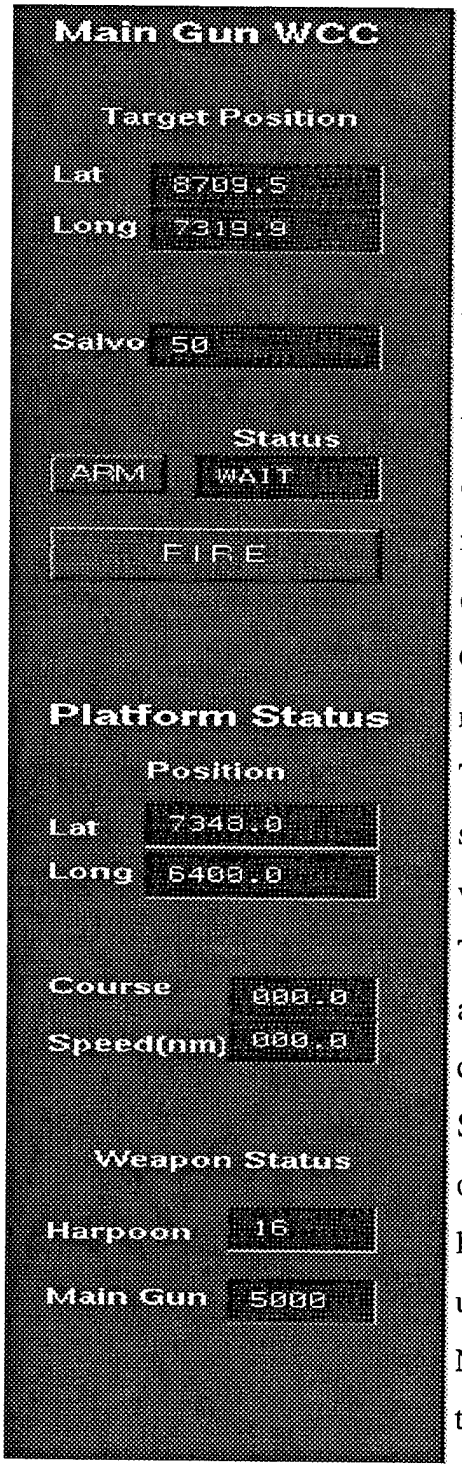


Figure 13: Gun/Platform Status

D. GUN ENGAGEMENT/ OWN SHIP PARAMETERS

The right hand side of the weapons control console is divided into two distinct areas. The upper portion is devoted to the information required for engagement of a target with shipboard guns. Its functions are similar to that of the missile engagement section. A number of salvos can be entered representing the number of rounds to send down range. Target position determines the slew and elevation of the gun. Communicate is conducted through shared memory.

The lower portion contains the true location of the ship within the world. As with other position windows it is shown as a latitude and a longitude. These however are misnomers. The latitude is actually the exact Y position of the ship in world coordinates. Similarly, longitude represents X. Since within the world X and Y positions can take on negative values, so too can the latitude and longitude displayed in position windows. This is unrealistic but it keeps in tune with the current NPSNET coordinate implementation. As long as the origin is correlated to the southwest corner of a flat world no problems should arise.

Along with the true position, own ships course and speed are displayed. These are critical pieces of information, especially during engagement planning stages. Figure 13 depicts this area of the Weapon Control Console.

E. SUMMARY

The Weapons Control Console is a stand-alone program that provides the required interface to plan weapons engagements. With the tri-fold function of radar display, engagement planning, and weapon launch, the WCC closely approximates fleet control consoles.

VIII. WEAPON IMPLEMENTATION

A. BACKGROUND

After creating a weapons interface suitable for use within NPSNET, consideration was given to the development of a shipboard weapon that more closely resembled those in use in the fleet. Current weapons in NPSNET do not have certain characteristics implemented that are unique to naval munitions, for example waypoint flight, radar seek, and vertical launch. It was decided to simulate a surface to surface (SSM) attack missile to demonstrate the use of the CICVET. Dubbed a Harpoon in the code and at the WCC, the missile developed only nominally represents the flight and attack profiles of one of the world's most commonly found SSMs.

B. NPSNET MUNITIONS CLASS

Previous work in NPSNET has created a base munitions class upon which various types of weapons can be built. Basic weapons implemented were bullets, bombs and missiles. These are essentially "straight-line" weapons in that they are targeted using visual means (a HUD cross hairs). Weapons are fired by depressing certain buttons either on the fire control joystick or on the keyboard. NPSNET handles these interrupts by instantiating the appropriate class of munition and invoking its member functions. The instantiated object becomes an entity within the simulation, updated similarly to the other entities.

The Harpoon missile was implemented as a derived class of the munitions class. This research focused on altering two of the member functions of the munitions class, the *move()* and *sendfire()*, and implemented a new function *seek()*. Additionally, changes were made to the launch effect animations and to the missile camera view.

C. HARPOON WEAPON CLASS

1. Launch

Because the actual launch command comes from a separate process, the weapon control console, a different approach is used for the instantiation of Harpoon missile objects. Lacking a direct interrupt to handle launch conditions, throughout the simulation process a check must be made to the shared memory area to catch the launch signal from the interface process (details of this area can be found in Chapter VII). Thus, instead of handling an interrupt through a callback function, the *sendfire()* function of a newly created Harpoon is called directly. This function creates the required packets to inform the network of a launch.

The source for the parameters to *sendfire()* is also different. This is to reflect the indirect launch methods used. In other words, the launch condition must flow from a stand-alone program executed by a NPSNET human entity. This stand-alone interface communicates to code that is driving a human entity. Since this entity has no inherent weapons, the code must establish which entity the human is mounted on in order to launch properly. Previous weapons received their data, i.e. position, orientation, and target simply by grabbing the data of the driven entity. In the case of the Harpoon, the mounted ship entity's data is utilized.

The shared memory is used not only for the launch condition but also the flight information. More data is required by the constructor when Harpoons are created than are required during instantiation of other weapons. This data comes from the missile engagement sequences that an operator inputs using the weapon control console. This data also replaces the line-of-sight information used by other weapons. Now the missile is "programmed" with its flight path prior to launch, mimicking the real world where surface-to-surface missiles are launched as 'fire-and-forget" munitions.

2. Flight

The flight of the missile entity is controlled by the *move()* member function. Within the structure of the Harpoon class four position values are maintained: three for waypoints and one for a final position. This final position is the target position, but it is pointed out that this position does not necessarily have a target located at it. It is simply the geographic position that a radar operator determined a possible target was at. The flight path is not based on the actual positions of any particular entity within the simulation. This distinguishes the Harpoon implementation from previous ones used by tanks and aircraft. The flight of the missile is controlled by these positions.

Waypoints are read from the shared memory area. First, the number of entered waypoints, one to three, is read. This enables the determination of the number of flight legs and when the seeker head should be activated. For each positional update, a comparison of the current position is made to the endpoint of the current leg. Using missile velocity the missile's position is advanced along the direction of the leg. Within a minimal distance from a leg endpoint, the missile is turned to the new leg. After the terminal leg is reached, the seeker head is enabled at a distance of 10,000 meters from the target position and guidance continues based on the *seek()* function.

3. Termination

When a distance of 10,000 meters from the final position is reached, the *seek()* member function is invoked in order to fine tune the target position. If the final point happens to be less than 10,000 meters away, then the seeker is invoked immediately upon the missile turning to the terminal leg. The seek() function uses the pfSegSet data member of the Harpoon class. This pfSegSet is created and used in much the same way as with the radar functions (see Chapter VI). In this case, the segments are positioned extending from the nose of the missile for a distance of 10,000 meters. The segments are swept side to side to simulate the onboard radar of the Harpoon.

When the missile is on its terminal leg and the seeker is enabled, the final position is used only if no intersection points are returned from the seeker head. When an intersection on a segment is successful, this point is used to correct the missile's flight. Adjustments are made to make this point the terminal one. The successful intersection test means that the missile will be driven toward an object and an impact will occur. It is pointed out that this intersection could be with any object in the world, a building, land or dynamic entity. Hopefully the information analyzed by the radar operator was good enough to produce a hit on a hostile entity. If no intersections resulted and the missile has reached its programmed destination, it self-destructs.

D. SUMMARY

A simulated surface-to-surface missile was implemented as part of the CICVET. Known as a Harpoon, it serves to enhance previous weapon implementation by augmenting capabilities. The Harpoon has waypoint flight, up to three, and a seeker head for terminal searching and target acquisition.

IX. CONCLUSION

A. RESULTS

The goal of this project was to implement weapons and sensor capabilities in the current NPSNET naval asset. In achieving this goal the following areas were explored;

- Large screen displays replace heads up display within the shipboard virtual environment. This method of information display/dissemination is more consistent with the real-world. LSDs are common in fleet combat information centers and are a better representation because real-world use expands a ship's interest domain through information links (i.e. satellite, radiowave).
- A functional radar was built. This radar uses intersection testing in an effort to simulate radar used in the fleet. With enhancements to ensure detection of dynamic objects, the radar can relate a two-dimensional picture of the world suitable for both weapons engagement and radar navigation.
- A Weapons Control Panel was constructed for engagement planning and weapons launch. A stand-alone program, the WCC communicates with NPSNET through a shared data-pool. It provides display capabilities to facilitate the planning stages of hostile encounters. Additionally, it provides the interface to the weapon entities within NPSNET.
- A new class of weapon was introduced to NPSNET. Nominally representative of the Harpoon surface-to-surface missile, the NPSNET Harpoon has inherent waypoint capability and a self search mode for target acquisition and destruction
- All CIC functions are executed only when inside CIC. This prevents any decrease in performance of NPSNET outside of the CIC.

The above additions to NPSNET create the foundation of a Combat Information Center Virtual Environment. By no means all-inclusive, it does demonstrate that indeed CICs can be modeled in networked virtual environs. As a integral part of a fully functional

shipboard VE, the CICVET could play a major role in supplemental training of navy personnel.

B. RECOMMENDATIONS FOR FUTURE WORK

This thesis was a continuation of work previously conducted in NPSNET. Although this work and others have produced significant results in the creation of a shipboard virtual environment, some areas of research remain unexplored. Below is a summary of some of these areas as they relate to CIC and the shipboard VE in general.

1. NPSNET V

Currently in the design stages, NPSNET V will break from the past and establish itself as a technological stand-out. We hope that special consideration will be paid to the unique aspects that naval assets have over other battlefield entities. These include longer range sensors, longer range weapons, and larger domains of interest. Taking into account how actual information is generated and utilized will be key. An implementation of radar is just one example.

2. Radar Enhancements

The radar of the CICVET was constructed from the beginning as a "can-we-do-it" project. To this end it does not reflect actual operating characteristics of any particular radar. Certain computational constraints in performing the necessary intersections preclude efficient radar implementation. These are being addressed in new releases of software toolkits such as Cosmo3D. Thus future implementations of intersection radar may be more feasible.

3. Improved Interface

A weapon control console as a stand alone program is not the ideal method. In fact it precludes one of the major reasons for using virtual environments. If the WCC could be implemented within the world then other players within the CIC could gain the information simply by glancing at the console in their view. At the time of this research, the creation of

a working console within the world adversely affected the frame rate so much as to prevent any useful information display. Such a console would naturally be built using the flexibility of direct OpenGL calls. Creating the geometry of buttons and knobs and then the capability for picking them is the reason for the computational increase.

4. Physically Based Weapons Suite Development for Ships

Follow on work could entail the physical modeling of real weapons suites. The Harpoon class of the CICVET merely touches the surface of today's weapons capabilities. In the future raw missile data could be fed into 3D simulations to get true to life fighting capabilities amongst entities.

5. Test and Evaluation of Training within a Virtual Environment

While establishing the foundation of a trainer for CIC personnel, the CICVET requires further enhancement and expansion to include the simulation of multiple consoles. After this, design and implementation of test and evaluation procedures for the exercises performed within the virtual environment must be created. Only after a well thought out and executed objective evaluation can one begin to discuss the merits or validity of VE training.

APPENDIX USER'S GUIDE

This appendix is the user guide for operating the NPSNET Ship Project System (NPS SHIP) and the radar/weapon control console. It explains how to start and operate the ship, human entities and then bring up the radar console. It does not cover the features of NPSNET; the operator should use NPSNET's User Guide located on the Naval Postgraduate School Computer Science Department's world wide web page (http://www-npsnet.cs.nps.navy.mil/npsnet) for further details.

A. STARTING NPS SHIP AND HUMAN ENTITY

NPS SHIP is a simulation run within the NPSNET vehicle simulator. NPSNET is a robust system capable of configuring many input and output devices to control vehicles in a simulation. NPSNET is the visual simulation manager of NPS SHIP. The user needs to have access to more than one workstation on a network, before starting NPSNET. To run NPSNETSHIP the user needs to go into ~npsnetIV/npsnet on workstation#0, and then into the configuration management view *eakyuz* and change to the directory /cm/npsnet on workstation#1 to start a weapons control console. The user should then execute the commands below. Because NPSNET is a robust system, it can be started in many different configurations..

Workstation #0 - start the ship entity
machine%bin/npsnetIV -f config.ship

Workstation #1 - start the human entity
[eakyuz]machine% bin/npsnetIV -f config.sailor

The "sailor" humans move within the virtual environment also using the standard NPSNET commands. The exceptions occur when transporting about the "antares96" ship, and when manipulating objects in the "antares96" ship. Once mounted on the "antares96"

ship, the "sailor" can transport to different locations about the ship using the keyboard commands described in Table A-2. To bring up the Weapons Control Console, the sailor

Transport Keys & Locations (you must be mounted on the ship to use these)	
F9	Pier on Island
CNTL F9	Cargo Deck/Vehicle Ramp
CNTL F10	Port Bridge Wing
CNTL F11	Engine Room Lower Level
CNTL F12	Combat Information Center

Table 2: Transport Keys & Locations

should go into Combat Information Center by pressing CNTL F12 upon being mounted to the ship entity or walk to the CIC. Additionally, the user can manipulate objects in the ship as follows. First, the simulator must be in the SHIP PICK mode. Using the middle mouse button, the user can change the NPSNET picking mode (see NPSNET User Guide for more information). Second, in order to manipulate an object in the model (i.e. picking up a nozzle), use the look mechanism (white arrow keys on key pad, or hats of the fcs) and line up the center cross hairs of the heads up display (HUD) or the center of simulation window over the object to be manipulated. Then press either the Left Button or Right Button as described in Table A-3 and Table A-4.

Joystick Buttons for Ship Picking	
Throttle #7	Left Button
Throttle #3	Nozzle Toggle Switch
Stick Bottom	Right Button

Table 3: Joystick Buttons for Ship Picking

Mouse Buttons for Ship Picking	
Left	Left Button
Middle	Change Picking Modes
Right	Right Button

Table 4: Mouse Buttons for Ship Picking

When the user manipulates a fixed object (such as bulkhead, deck, ceiling, desk, etc.), information about that object is displayed in the unix shell window. When the user manipulates a movable or selectable object (such as doors, valves, or Weapons Control Consoles), the movable object moves in the direction of the buttons, either opening or closing, If it is a selectable object like Weapons Control Console another window pops up and the weapons control console program runs in it. Another way to run the weapons console is to go into the CIC and execute the following command in *eakyuz* view.

```
Workstation #1 -
[eakyuz]machine% bin/radar
```

B. OPERATION PROCEDURE OF WEAPON CONTROL CONSOLE

Once NPSNET and Weapon Control Console run, communication between two processes is established via a shared data pool. The Weapons Control Console starts to draw environmental information upon reading it from datapool and starts to write radar range information to the datapool for use by NPSNET. This communication continues during execution for further information exchange purposes, like weapon engagement data from weapon control console and NPSNET. General screenshot of Weapons Control Console is giving in Figure 14. Detailed command buttons and information boxes are explained in following sections of the user guide.

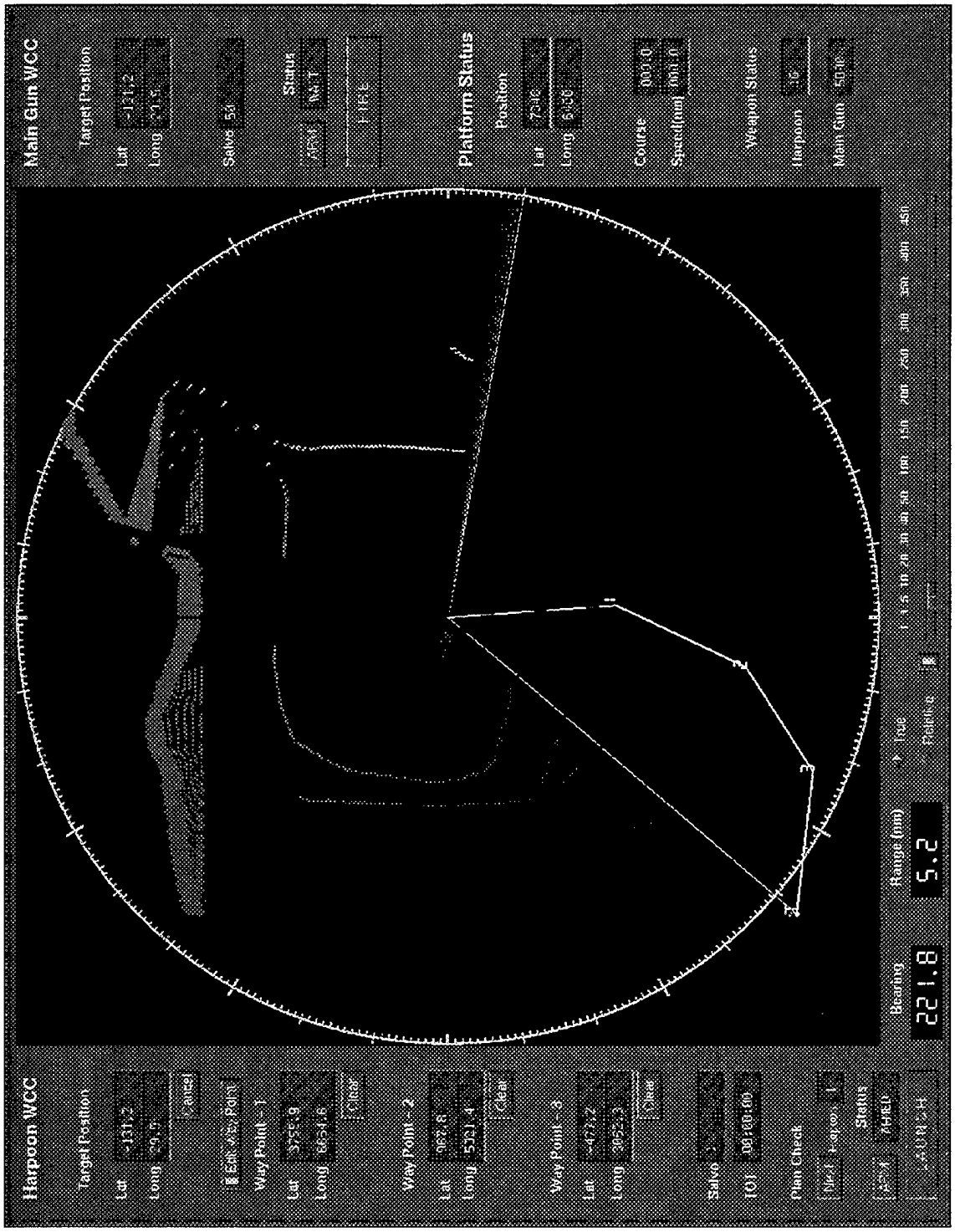
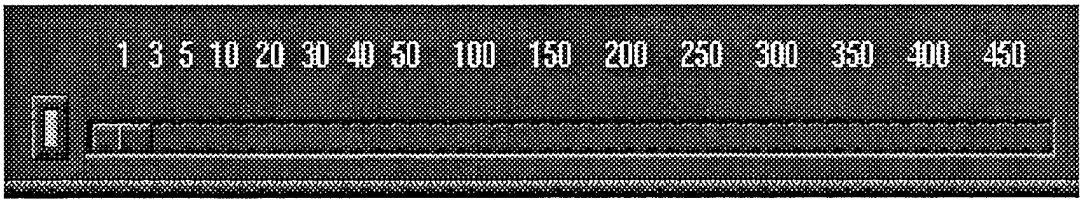


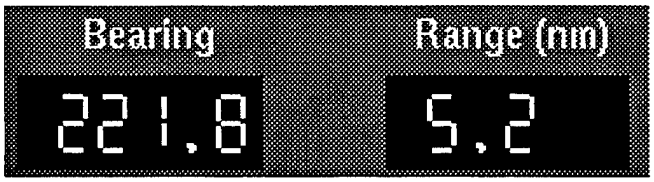
Figure 14: Weapons Control Console screen shot

1. Radar Range Scale:



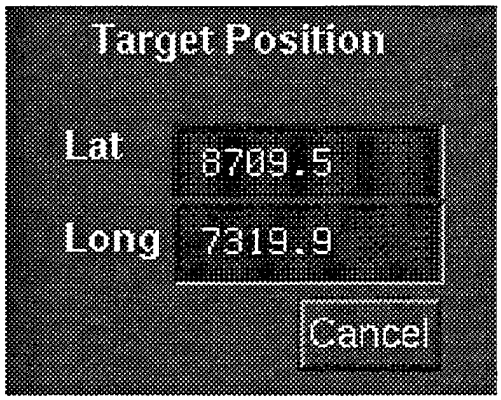
The displayable range of the radar is selectable by a slide bar. For the first 50-mile, range rings are drawn at an interval of 10 miles, after that they are drawn every 50 miles. Range rings are turned on as default and can be turned off by the toggle switch located at the left hand side of the range scale slide bar. The radar range set by the Weapons Control Console is written to a datapool to be read by NPSNET. Range rings and bearing indicators on the outer circle are implemented to give fast location information to the operator.

2. Bearing and Range Indicators:



Bearing and Range indicators shows the current position of the radar cursor which is controlled by the left mouse button. It can be used to check a specific range or to get the true position of a track on the radar display. They indicate the true bearing (from true north) in degree and the range in nautical miles.

3. Target Position:



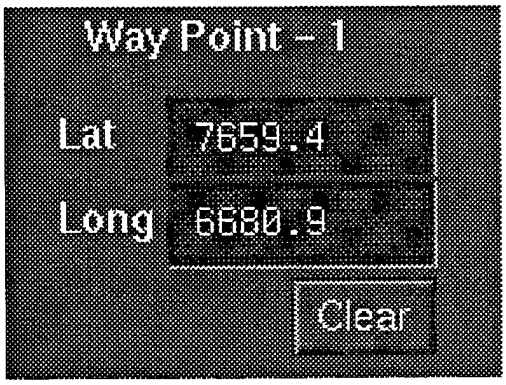
Target position information is invoked by clicking a location on radar screen with the right mouse button. This will also place a letter "T" to indicate the selected location as a target and draws a line connecting a platform's position to target's location to simulate the missile path. This information is fed into both the target position windows of missile engagement and main gun sections. The Cancel button under the harpoon control console is used to cease an engagement for a specific launcher, and clears the launching information from its local storage. Target position can also be entered or corrected by using the system keyboard.

4. Edit waypoint:

This button selects between edit and display modes of waypoint windows and radar display. In edit mode, waypoint windows are used to display the tentative waypoint information. Once the "ARM", which is explained below, is pressed this information is stored in local storage and becomes fixed until it is cleared by the button below it. In display

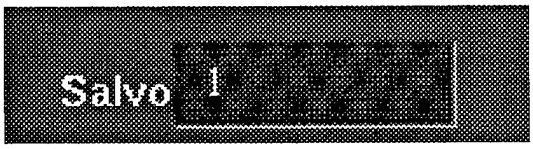
mode, the waypoint window displays the location information for that waypoint and in this mode it can not be cleared.

5. waypoint - 1, 2, 3:



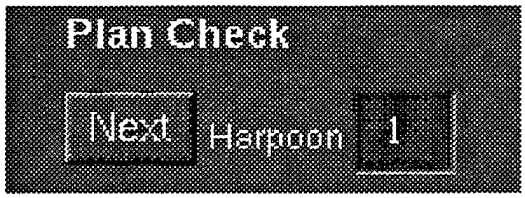
These windows display the position information in world coordinates for the waypoints. This information is input into the windows by clicking on the desired waypoint locations with the middle mouse button, or they also can be entered or corrected manually by way of keyboard. They can be edited in "waypoint edit mode" prior to launch even after being stored with "ARM" button.

6. Salvo:



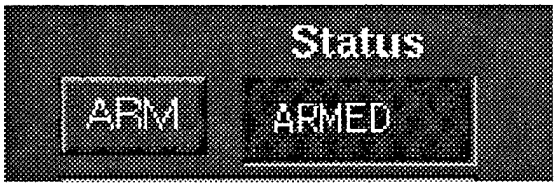
This window displays the total number of missiles armed and ready to launch. Following a launch it zeros itself automatically. For main gun operation it indicates the number of rounds armed for the current target, and for the main gun it needs to be entered manually.

7. Plan Check:



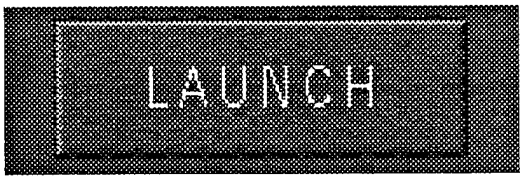
This area of the control panel is used for two different purposes. In waypoint edit mode it is used to input target and waypoint information for more than a single missile. Pressing this button will cycle to the next missile cell. In edit mode, it maintains the information in waypoint fields so that flight paths can be copied from missile to missile. In waypoint display mode it displays the information of the missiles one by one upon selection. Missile paths will also be drawn into the display.

8. Status:



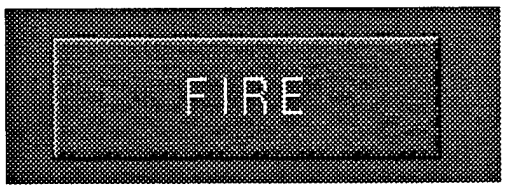
This field displays the status of the current missile cell selected. Possible statuses for a missile are "READY", "ARMED", and "EMPTY", for a main gun there is no "EMPTY" status. Default status for all missiles is "READY", upon selection of the "ARM" button next to this field, all target and waypoint information is stored into the missile structure and its status is changed to "ARMED". When a missile is launched its status is changed to "EMPTY", and this last status will prevent the missile from being launched more than once.

9. Launch:



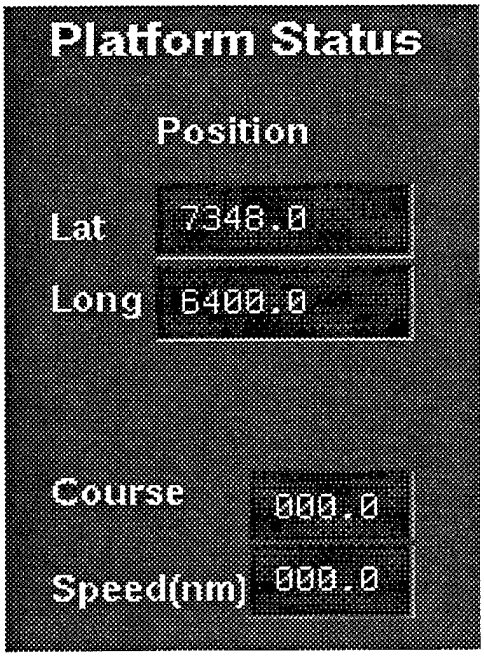
Launch button writes the target and waypoints information into the shared datapool and signals NPSNET to launch. Upon acknowledgment from NPSNET it sets empty cells and also decreases the number of available missiles by the number of missiles launched in this salvo.

10.Fire:



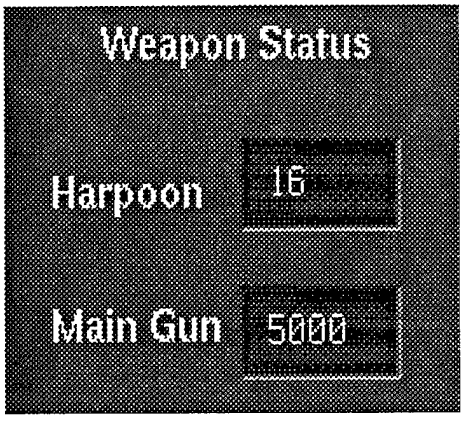
This button does the same thing as the launch button does for the missile part of the control panel. It writes the target information into the shared datapool, decreases the number of available rounds by the salvo number, and changes status from "ARMED" to "READY".

11.Platform Status:



This status field displays the course, speed, and location information of the platform which the radar is running on. It obtains the information from the shared datapool which is written by NPSNET.

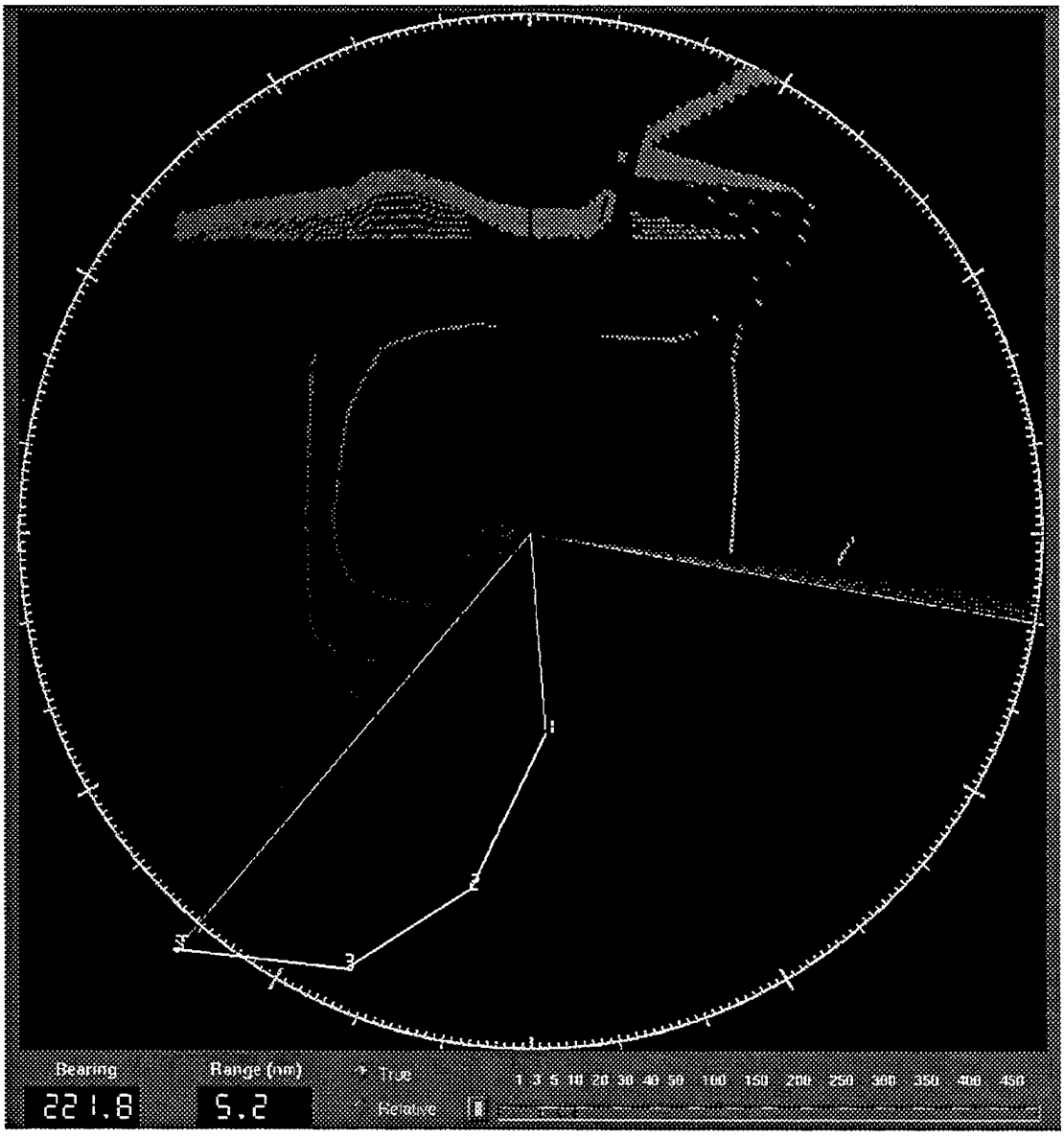
12.Weapon Status:



These two windows indicate the available ammunition units for the harpoon guided missile and the main gun for the platform. Those amounts are preset to 16 for the harpoon guided missile and 5000 for the main gun ammunition. When launch is established from its

weapons control panel, the available number of missiles and gun rounds are dropped by the amount of the salvos launched/fired in this engagement.

13.Radar Window:



Radar window displays the environmental information gathered and passed by the NPSNET, it draws the all information as raw data. Final evaluation and classification of the

data is left to the user. It is also being used to draw guided missile attack paths, and range rings.

LIST OF REFERENCES

[HITL94] Human Interface Technology Laboratory, Washington Technology Center, University of Washington, "A Crisis Management Testbed for Experimental Training of Damage Control Assistants.", March 31, 1994.

[IRIS95] Silicon Graphics, Inc., IRIS Performer Programme's Guide, Software Manual, 1995.

[LOCK94] Locke, John, "An Introduction to the Internet Networking Environment and SIMNET/DIS.", Computer Science Department, Naval Postgraduate School, Monterey, California, October 24, 1994.

[MCDO95] McDowell, Perry and King, Tony, "A Networked Virtual Environment for Shipboard Training.", Master's Thesis, Naval Postgraduate School, Monterey, California, September, 1993.

[MULT95] MultiGen Inc., MultiGen Modeler's Guide, Software Maual, MultiGen Inc., 1995.

[NOBL95] Nobles, Joseph Anthony, and Garrova, James Francis "Design and Implementation of Real-Time, Deployable Three Dimensional Shiphandling Training Simulator" Master's Thesis, Naval Postgraduate School, Monterey, California, June 1995.

[OBYR95] O'Byrne, James E. "Human Interaction within a Virtual Environment for Shipboard Training.", Master's Thesis, Naval Postgraduate School, Monterey, California, September 1995.

[OPEN93] OpenGL Architecture Review Board, OpenGL Programming Guide, Software Manual, Silicon Graphics, Inc., 1993.

[OSBG92] Osberg, K. Virtual Reality and Education: A Look at Both Sides of the Sword. HITL Report. No.R-93-7. (HTML Document).

[RAPI95] Silicon Graphics, Inc., Developers Magic : Rapidapp User's Guide, Software Manuel, Silicon Graphics, Inc., 1995.

[STEW96] Stewart, Bryan Christopher "Mounting Human Entities to Control and Interact with Networked Ship Entities in a Virtual Environment.", Master's Thesis, Naval Postgraduate School, Monterey, California, March 1996.

[ZESW93] Zeswitz, Steven R., "NPSNET: Integration of Distributed Interactive Simulation (DIS) Protocol for Communication Architecture and Information Exchange." Master's Thesis, Naval Postgraduate School, Monterey, California, September, 1993.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. Chairman, Code CS 2
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943-5000

4. Dr. Michael J. Zyda, Professor 2
   Computer Science Department Code CS/ZK
   Naval Postgraduate School
   Monterey, CA 93943-5000

5. John S. Falby, Senior Lecturer 2
   Computer Science Department Code CS/FA
   Naval Postgraduate School
   Monterey, CA 93943-5000

6. LT John J. Kapp 2
   1434 Bryn Mawr St.
   Scranton, PA 18504

7. LT(jg) Erkan Akyuz 2
   Acibadem Cad. Yeniyol
   Kocaturk Sit. A/12
   Kadikoy, Istanbul 81010
   Turkey

8. Deniz Kuvvetleri Komutanligi 2
   Personel Tedarik ve Yetistirme Daire Baskanligi
   Bakanliklar, Ankara 06100
   Turkey

9. ECJ6-NP 1
HQ USEUCOM
Unit 30400 Box 1000
APO AE 09128